J. Int. Financ. Markets Inst. Money 105 (2025) 102210

Contents lists available at ScienceDirect

Journal of International Financial Markets, Institutions & Money

journal homepage: www.elsevier.com/locate/intfin

Machine learning, memory and efficiency in cryptocurrency markets

Shuyue Li, Larisa Yarovaya *, Tapas Mishra

University of Southampton Business School, University of Southampton, United Kingdom

ARTICLE INFO

JEL classification:
C22
C45
G10
G15

Keywords:
Cryptocurrency
Machine learning
Seq2Seq
Long memory

ABSTRACT

This paper empirically examines whether machine learning (ML) methods can capture long memory in the cryptocurrency markets. We design two tests to evaluate seven widely used ML regression algorithms and sequence-to-sequence (Seq2Seq) models to determine their ability to capture long-memory characteristics of financial data. Specifically, we assess their accuracy in estimating the fractional integration parameter d for both univariate and systemic memory. Additionally, we examine whether the predicted time series preserve the long-memory properties of the original cryptocurrency market data. Our findings reveal that most ML algorithms fail to handle long-memory series effectively, while models incorporating Long Short-Term Memory (LSTM) and Attention-LSTM components exhibit superior performance. Whilst comparing models using Mean Squared Error (MSE), we find that our tests identify models better for directional predictions. These results highlight the limitations of conventional ML mechanisms for long-range dependence and position Seq2Seq models as a promising alternative for addressing the complex movements of cryptocurrency time series. Our approach can be readily extended, offering both academics and practitioners a systematic procedure for evaluating arbitrary ML models, thereby yielding insights not only into their generalization performance but also into the interpretability of their capacity to model long-term dependence.

1. Introduction

With the rapid advancement of both blockchain and artificial intelligence technologies, a proliferation of models has been developed for price prediction in cryptocurrency markets, creating substantial challenges for model selection. Traditionally, MSE has served as a fundamental criterion for evaluating predictive performance. However, from a practical standpoint, reliance solely on MSE may be insufficient for identifying models that effectively capture complex data structures necessary for robust trading strategies. To address this limitation, we introduce two novel tests designed to assess AI models' capacity to capture long-term dependence, thereby facilitating the selection of models suited for time series exhibiting long memory.

Long memory is instrumental in assessing the degree of market inefficiency and can be rigorously quantified through the fractional integration method (Peters, 1994). The fractional integration parameter d provides crucial information regarding trending, mean reversion, and random-walk characteristics within price series, enriching our understanding of market dynamics. Recent studies by Hassler (2018), Duan et al. (2021), and Mishra et al. (2023) systematically examine the relationship among the values of d, market efficiency, and price dynamics. Building upon their findings, Table 1 summarizes how different values of d correspond to distinct regimes of price behavior and market efficiency.

* Corresponding author.
E-mail address: l.yarovaya@soton.ac.uk (L. Yarovaya).

https://doi.org/10.1016/j.intfin.2025.102210
Received 6 December 2024; Accepted 19 August 2025
Available online 6 October 2025

Table 1
Illustrations of the relationship between memory and market efficiency for log-return series.

Interval of d	Memory	Market efficiency	Price dynamics	Degree of mean reversion
$0.5 \leq d < 1$	Long memory	Inefficiency	Strong Trend	None
$0 < d < 0.5$	Long memory	Inefficiency	Persistent Trend	Weak
$d = 0$	Short memory	Efficiency	Random Walk	None
$-0.5 < d < 0$	Short memory	Inefficiency	Mean Reversion	Strong

Note: According to Caporale et al. (2018), the differencing parameter d of the univariate series is related to the Hurst exponent described above through the relationship $H = d + 0.5$.

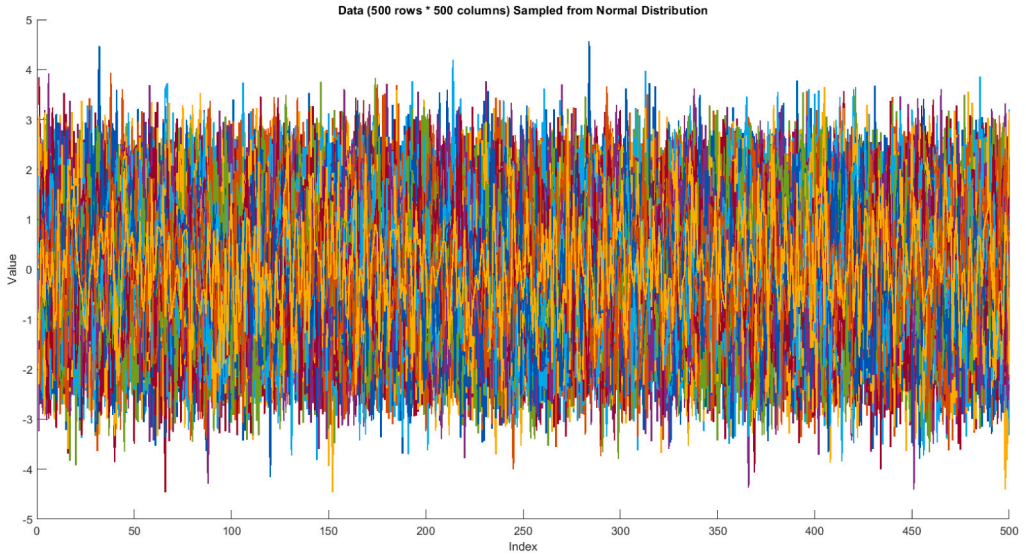


Fig. 1. Simple white noise (d=0).
Note: Fig. 1 presents noise data sampled from a normal distribution, representing the log returns of assets consistent with the EMH by Fama (1970). This data matrix comprises 500 rows and 500 columns. We apply fractional differencing to each column to simulate series exhibiting diverse memory characteristics, specifically with memory parameters $d = -0.7, -0.3, 0, 0.3, 0.7$.

Unlike traditional financial assets, cryptocurrencies lack intrinsic value anchors such as dividends or cash flows. They are traded in markets characterized by high volatility, minimal regulatory oversight (Diniz-Maganini et al., 2021), heterogeneous market participants (Corbet et al., 2019), and time-varying herding behaviors (Yarovaya et al., 2021). These distinct attributes contribute to persistent inefficiencies in cryptocurrency markets, where prices may not reflect fundamental values, as shown by Cheah and Fry (2015), the fundamental value is zero, and, as a result, price dynamics can be largely driven by noise, including market manipulation, information asymmetry, investor sentiment, and event-driven shocks. From this angle, financial markets can be inefficient if useful information can be extracted from the noise (Urquhart, 2016; Hirano et al., 2018; Wei, 2018; Bundi and Wildi, 2019; Kristoufek and Vosvrda, 2019). However, a gradual shift towards efficiency has been observed in the cryptocurrency market over time, especially from 2017 onwards, suggesting that the efficiency of the cryptocurrency market is adapting and evolving, possibly due to increased market participation and improved market structures (Hu et al., 2019; Le Tran and Leirvik, 2020; Noda, 2021; Duan et al., 2023). In fact, the fractal analysis has been applied to Bitcoin markets by Grobys et al. (2022), Grobys (2023), highlighting the presence of fractal properties of cryptocurrency data. Figs. 1 through 6 visually illustrate how different degrees of market efficiency affect price and return dynamics and the tails of their distributions. Thus, investigating whether machine learning methods can effectively capture long-memory patterns is of significant importance for cryptocurrency price prediction. In particular, although gradient descent is the standard approach for training deep learning models, learning long-term dependence with this approach is difficult (Bengio et al., 1994). Moreover, Zhao et al. (2020) argue that recurrent neural networks (RNN) and LSTM models struggle to replicate the statistical long-memory characteristics.

In this paper, we design two rigorous and universal tests to examine the conventional ML and Seq2Seq models. The selection of these models is motivated by their popularity and availability, as well as their demonstrated performance in prior literature. ML models include linear models, multilayer perceptron (MLP), random forest (RF), regression tree (RT), support vector regression (SVR), Gaussian kernel regression (GKR) and Gaussian process (GP) regression. The Seq2Seq models contain LSTM, Bidirectional LSTM (BiLSTM), and Gated Recurrent Unit (GRU) with attention components. The first method aims to investigate whether the predicted series retain long-memory properties. The second method aims to test whether these powerful models can estimate the fractional integration parameter d of the series. These two tractable tests contribute to the critical evaluation of the prevalent AI models applied to financial data from an innovative perspective centered on the consistency of memory properties, emphasizing their

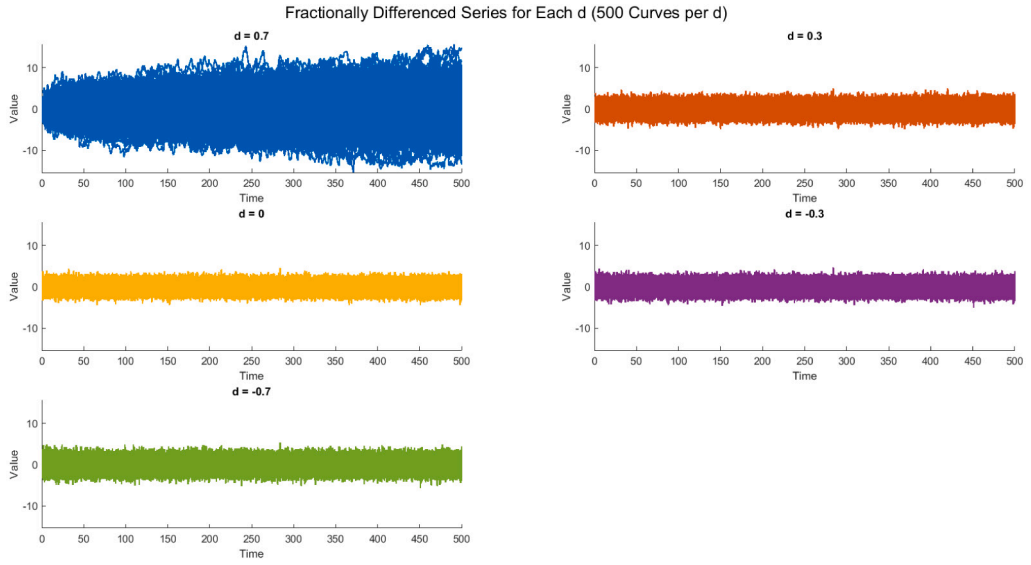


Fig. 2. Simple white noise (d=0) and Fractional noise with different values of d.
Note: Fig. 2 depicts fractional noise behaviors, highlighting that the series with $d = 0.7$ exhibits significantly increased volatility, indicating a high degree of market inefficiency. In contrast, series with $d < 0.5$ behave similarly to the original series with $d = 0$.

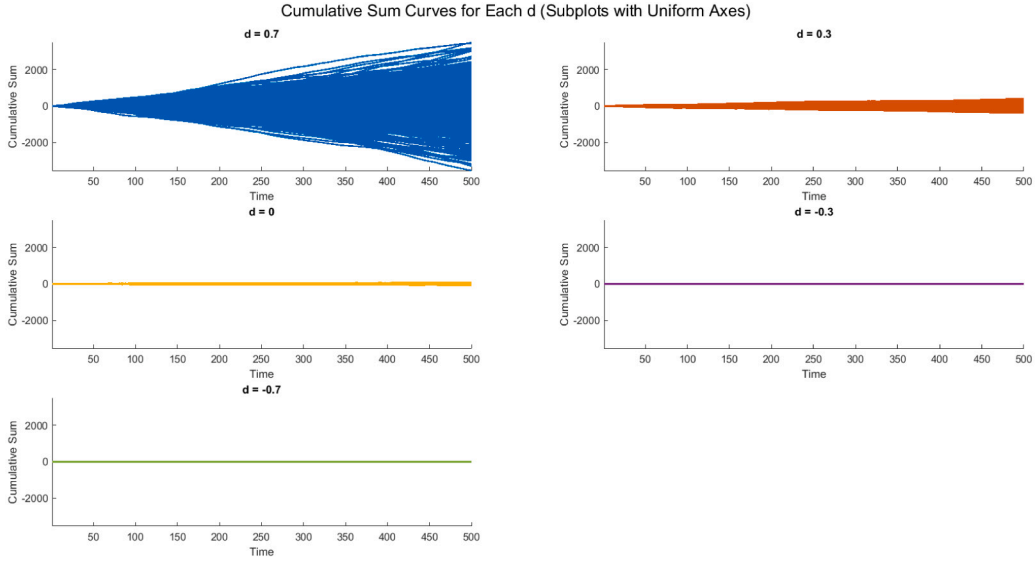


Fig. 3. Cumulative sum curves for different d.
Note: Fig. 3, representing the cumulative sum of white noise and fractional noise, illustrates clear trending behavior for series with $d > 0$, whereas series with $d < 0$ display mean reversion.

importance in financial modeling. Specifically, this paper provides theoretical, methodological, empirical and practical contributions to the financial markets literature.

From a theoretical perspective, we innovatively provide interpretability for varying performance of ML models in financial markets from a long-memory perspective (Ahmed et al., 2010; Sermpinis et al., 2014; Fischer et al., 2018; Siami-Namini et al., 2018; Rundo et al., 2019; Ryll and Seidens, 2019; Aygun and Gunay, 2021; Tang et al., 2022; Alexandridis et al., 2024; Huang et al., 2024; Zhang et al., 2024). This contribution advances the current literature by highlighting the need for a more integrated understanding of statistical characteristics, within the context of ML model applications. We enrich existing comparative studies on statistical and ML models in financial markets and offer new insights into how model performance is inherently linked to the statistical properties of the data. Our findings underscore the critical importance of incorporating statistical consistency into the ML modeling process to enhance robustness and interpretability in financial forecasting. Furthermore, this procedure can be extended

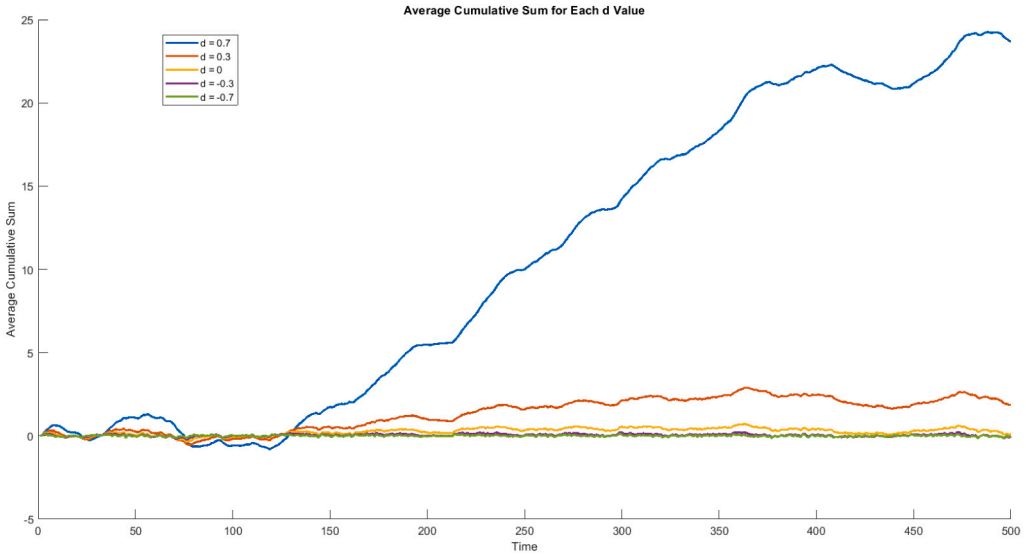


Fig. 4. Average Cumulative Sum under different values of d.
Note: Fig. 4 demonstrates the average trajectory of 500 simulated series with various memory parameters, clearly highlighting either persistent trends or mean-reverting patterns.

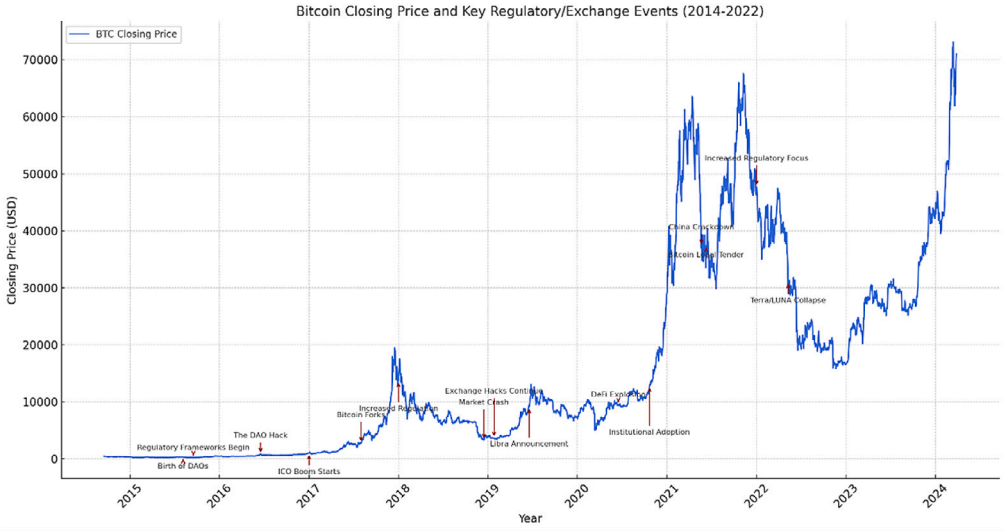


Fig. 5. Bitcoin Closing Price.
Note: Fig. 5 illustrates Bitcoin price dynamics, underscoring considerable market inefficiency characterized by persistent trending behavior, indicative of long-memory dynamics.

into a comprehensive framework for machine learning models to examine key statistical properties, such as kurtosis and skewness. As more tests on the generated series are conducted, the generalization of machine learning applications in financial contexts can be significantly enhanced.

From a methodological perspective, we propose an innovative and universal method to assess whether advanced models can generate long-memory series, filling a notable gap in the study by Mullainathan and Spiess (2017). To our knowledge, this is the first paper to evaluate the effectiveness with which Seq2Seq models learn fractionally cointegrated series, shedding new light on the ability of these models to capture complex time dependence in financial markets. This method not only supports the development of more sophisticated models, but also ensures the generalization of such models across various financial datasets, making a significant contribution to the ongoing debate on ML model validation and applicability. Moreover, these methods also contribute to the development of estimators for the values of d, thereby facilitating the assessment of market efficiency through ML algorithms.

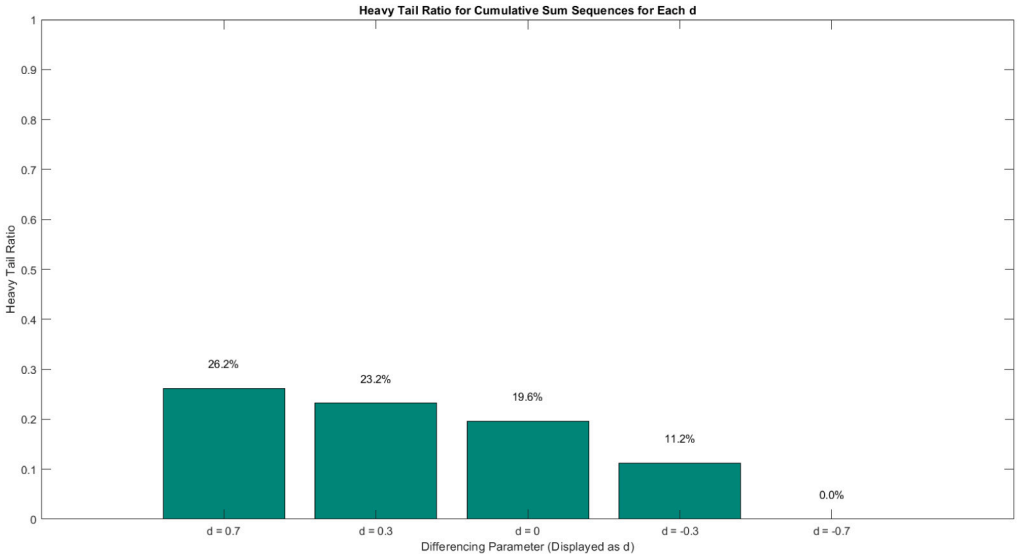


Fig. 6. Percentage of Heavy Tail Distribution for Cumulative Sum Sequences with various d.
Note: Fig. 6 focuses on the implications of long-memory returns on price series, demonstrating that increased inefficiency correlates positively with a higher proportion of heavy tails in price distributions. The degree of heavy-tailedness is estimated using the Hill estimator proposed by Hill (1975), which remains a widely adopted method for assessing the tail index of power-law distributed financial series. Given the inherent complexity and inefficiency observed in cryptocurrency markets, we conduct simulations across a variety of distributions to examine how varying degrees of inefficiency influence their statistical properties. The simulation results are presented in Figs. 22 through 33. In general, as market inefficiency increases, the heavy-tailed ratio of the cumulative return series also rises, a pattern consistent with the behavior observed under the Normal distribution. Additional details regarding the simulation procedures and results are provided in the Appendix section.

Empirically, we provide evidence that LSTM models can retain the long memory property in predicted series, echoing the findings of Boros et al. (2024), while contrasting with the results of Zhao et al. (2020), which suggested limitations in retaining long-term dependence. Our results further demonstrate that the Embedding-Attention-LSTM model outperforms other architectures, illustrating the potential effectiveness of hybrid models (Zheng et al., 2021). This empirically supports the findings of Ryll and Seidens (2019), which suggest that Seq2Seq models generally outperform traditional models. Furthermore, our findings contend that BiLSTM struggles with systemic long-term dependence, challenging the conclusions of Ubal et al. (2023), who argued for the superiority of BiLSTM models in predicting the Hurst exponent. Furthermore, we critically engage with the works of Ledesma-Orozco et al. (2011) and Mukherjee et al. (2023), who employed neural networks to estimate the Hurst exponent, and provide empirical evidence supporting the arguments of Lommers et al. (2021), emphasizing the importance of adjusting ML models to account for the idiosyncrasies and statistical properties of financial data. These empirical findings provide valuable insights into the ability of machine learning models to capture market efficiency and highlight the differences in their effectiveness at identifying inefficiencies across various financial time series. Furthermore, they offer guidance for selecting appropriate model architectures and techniques, particularly in the context of analyzing long-memory processes and improving forecasting performance.

Finally, from a practical perspective, our tests contribute to model selection and development based on the ability of models to handle long-memory series, offering crucial guidance in the process of selecting the most appropriate model for specific financial applications. In the context of the rapid proliferation of ML models, our work emphasizes the urgent need for a tractable method that not only evaluates model performance but also ensures interpretability. This helps researchers and traders choose suitable models for deployment in complex and dynamic financial environments. As we argue, the current landscape does not suffer from a lack of predictive models but rather from the absence of a systematic approach to choosing the right model for accurate and interpretable predictions. By providing these two tests, we fill this essential gap and pave the way for more informed model selection in financial markets.

The rest of the paper is organized as follows. Section 2 reviews the key literature. Section 3 discusses the main methods employed in this study. Section 4 describes the data and documents our empirical findings. Section 5 draws the conclusion of the paper. Section 6 is the appendix.

2. Literature review

The core theoretical underpinning of ML hinges on the assumption that observable data conforms to an identifiable underlying distribution and pattern that algorithms can systematically approximate. Through iterative exposure to training datasets, ML architectures progressively refine their parameter configurations to minimize deviations between predicted and observed outcomes. This calibrated optimization process enables subsequent out-of-sample forecasting applications. As demonstrated by Hornik et al.

(1989), even single hidden-layer multilayer perceptrons possess universal approximation capabilities, theoretically enabling them to emulate arbitrary continuous distribution functions, given sufficient network complexity. This mathematical property raises the possibility of ML models replicating sophisticated financial data-generating mechanisms, particularly in modeling nonlinear relationships that challenge conventional econometric approaches.

Nevertheless, critical uncertainties persist regarding ML's capacity to capture long-range dependence patterns inherent in financial time series. The architectural constraints of finite-memory neural networks, coupled with the exponential decay of gradient information in deep learning frameworks, pose significant challenges in learning persistent temporal relationships spanning extended horizons (Bengio et al., 1994). This uncertainty motivates our subsequent empirical investigation into ML architectures' temporal dependence learning capacities within complex financial forecasting environments.

2.1. Cryptocurrency distribution

A growing body of literature explores the statistical properties of cryptocurrency returns, with particular emphasis on distributional characteristics and tail behavior. Begušić et al. (2018) confirm the heavier tails of Bitcoin returns relative to equities, underscoring the prevalence of extreme price movements in cryptocurrency markets. Kakinaka and Umeno (2020) demonstrate that while the cubic power-law model better captures the extreme tails of return distributions, Lévy stable laws provide a superior fit across the full range of returns. This finding highlights a key trade-off between tail precision and overall distributional adequacy. In contrast, Núñez et al. (2019) advocate for the normal inverse Gaussian (NIG) distribution in modeling BTC returns, citing its ability to accommodate both heavy tails and bubble episodes. The NIG's closure under convolution further enhances its applicability for multivariate risk assessment, with performance comparable to generalized hyperbolic (GH) models in out-of-sample evaluations. Extending this line of inquiry, Kończal and Wronka (2024) examine BTC, ETH, DJIA, and HSI markets, finding that NIG distributions offer universal applicability, whereas the double Weibull and Student's t distributions yield better fits for specific assets. Complementary findings by Shanaev and Ghimire (2021), based on a comprehensive evaluation of 22 distributions across 772 cryptocurrencies, indicate that Johnson SU and asymmetric power functions consistently outperform other candidates, especially for large-cap cryptocurrencies.

From a dynamic perspective, López-Martín et al. (2021) underscore the advantages of skewed distributions, particularly SGED, although there is no single static distribution that systematically fits the data better. Notably, Ouandlous et al. (2022) observe that Bitcoin's tail exponent post-2015 exceeds two, moving beyond the Lévy stable regime and indicating that distributions are time-varying. These studies highlight the complexity and heterogeneity inherent in modeling cryptocurrency returns. While heavy-tailed and skewed distributions are essential for capturing extreme events and asymmetries, no single model consistently provides a universally superior fit. The divergence in optimal distributional choices across assets and time periods highlights the need for dynamic selection or a model-free approach to fit real-world distributions. In this regard, ML, as a set of data-driven methodologies and universal function approximators, may offer promising avenues for modeling such time-varying distributions. Moreover, these findings raise important considerations regarding market maturity and the evolving statistical characteristics of digital assets, particularly as cryptocurrencies transition from speculative instruments to more established financial products. A notable stylized fact is the frequent co-occurrence of heavy tails with long memory and market inefficiencies. However, the interaction between long-term memory dynamics and heavy-tailed behavior remains insufficiently explored. This paper aims to contribute to addressing this gap by examining the potential linkages between memory and the tail properties of distributions.

2.2. Fractional differencing, d estimation and machine learning

Given a univariate time series $\{X_t\}$, a fractional differencing equation can be written as:

$$\tilde{X}_t = \sum_{k=0}^{t} w_k X_{t-k}, \tag{1}$$

$$w_k = -w_{k-1}\frac{d-k+1}{k}, \tag{2}$$

where d is the order of fractional difference, $w_0 = 1$, $\tilde{X}_t$ is the weighted sum of past (lagged) values. As shown, this formula can literally capture long-term information from historical data related to the persistence and predictability properties of time series.

As financial econometrics has increasingly embraced more sophisticated frameworks, the introduction of the Fractionally Cointegrated Vector Autoregressive (FCVAR) model marks a significant advancement in the analysis of systemic memory. This model is particularly well-suited for capturing fractional cointegration, which reflects long-term equilibrium relationships, alongside short-term dynamics, especially between high and low price series (Baruník and Dvořáková, 2015; dos Santos Maciel and Ballini, 2019; Maciel, 2020; Yaya et al., 2022). By incorporating fractional integration orders, FCVAR models allow researchers to describe persistent dependence in non-stationary time series more accurately, thereby enhancing our understanding of price dynamics in virtual currency markets and extending the contributions of Ciaian et al. (2018) and Sapkota and Grobys (2021).

However, cointegration relationships are not necessarily constant over time, as emphasized by Huang et al. (2023). A key unresolved issue in the literature is whether fractional cointegration itself can undergo regime changes, implying that the memory parameter D may vary across different market conditions. Despite its importance, effective approaches to identify and model this potential time-variation remain scarce. To bridge this methodological gap, ML techniques offer a promising pathway for forecasting

the evolution of the memory parameter D, potentially contributing to the development of regime-switching or time-varying parameter FCVAR models.

The univariate memory parameter d is estimated by the Feasible Exact Local Whittle (FELW) estimators by Shimotsu (2010) and systemic memory parameter D is estimated by the Fractionally Cointegrated Exact local Whittle (FCELW) estimator by Shimotsu (2012). Recent studies indicate a growing trend towards employing ML models for estimating long memory, primarily due to their ability to overcome limitations inherent in traditional methods, such as graphical interpretation constraints and intensive computational demands. Specifically, Ledesma-Orozco et al. (2011), Mirzaei et al. (2014), and Mukherjee et al. (2023) utilize artificial neural networks (ANN) to estimate the Hurst parameter. Meanwhile, Boros et al. (2024) investigate the potential of LSTM networks in estimating the Hurst parameter within fractional stochastic processes. Ubal et al. (2023) demonstrate that BiLSTM networks outperform RNN, LSTM, and GRU models in predicting the Hurst exponent. Such ML-based approaches effectively capture and predict long memory behaviors, enhancing time series forecasting by adapting dynamically to evolving memory characteristics. These findings motivate us to systematically evaluate additional ML and deep learning methods for estimating the long memory parameter, d. Furthermore, we innovatively propose a test specifically designed for Seq2Seq models, aiming to evaluate the capability of these models to identify fractionally cointegrated relations.

Despite the promising outcomes reported in prior research, Zhao et al. (2020) have statistically demonstrated that RNN and LSTM models inherently exhibit short-memory characteristics. Zheng et al. (2021) further extend the literature by illustrating that the incorporation of attention mechanisms significantly enhances the long-term memory capabilities of LSTM networks. Liu et al. (2022) advance this perspective by proposing a hybrid model combining MLP, Feedforward Attention mechanisms, and LSTM networks to improve multivariate time series forecasting.

These studies collectively underscore the potential of Seq2Seq models in effectively capturing long-memory dynamics. Nevertheless, to the best of our knowledge, systemic empirical comparisons among these prevalent algorithms remain notably scarce. Additionally, while certain models have been identified for their superior performance, explanations regarding the underlying reasons for their comparative advantage are typically lacking. In response, our study seeks to provide explanatory insights from a long-memory perspective, addressing the critical gap concerning the statistical consistency of the myriad available models and their numerous variants.

2.3. Statistical models and machine learning algorithms for time series data prediction in financial markets

Both traditional statistical models and contemporary ML algorithms hold substantial importance within financial market research, prompting extensive investigation into their predictive capabilities.

Motivated by the review conducted by Alsharef et al. (2022a), forecasting methodologies can be classified into five distinct categories: linear econometric models (e.g., the ARIMA family); non-linear econometric models (such as the GARCH family); regime-switching and threshold models; traditional ML algorithms optimized by AutoML procedures, including SVR, RF, RT, MLP, GKR, GP; and deep learning models encompassing Deep Neural Networks (DNN), RNN, LSTM, BiLSTM, GRU, Attention mechanisms, and hybrid approaches.

A subsequent empirical analysis by Alsharef et al. (2022b) reveals that deep learning methods consistently outperform econometric models in forecasting historical BTC and Ethereum (ETH) prices and BTC volatility (Huang et al., 2024). This finding aligns with existing literature that documents the superior predictive performance of deep learning methodologies (Siami-Namini et al., 2018; Rundo et al., 2019; Aygun and Gunay, 2021; Soni et al., 2022; Pirani et al., 2022; Chatterjee et al., 2021; Tang et al., 2022), extending across various forecasting horizons (Derbentsev et al., 2019) and the directional classification of price movements (McNally et al., 2018). Nevertheless, contrasting findings by Makridakis et al. (2018) suggest that statistical methods yield higher accuracy than ML techniques when applied to smaller datasets. However, this advantage diminishes with increasing dataset size, leading ML approaches to achieve superior forecasting performance over traditional statistical methods (Cerqueira et al., 2022).

Moreover, a substantial body of evidence demonstrates that traditional ML models frequently outperform conventional econometric approaches in various forecasting tasks (Mullainathan and Spiess, 2017; Sapankevych and Sankar, 2009; Hsu et al., 2016; Qian and Gao, 2017; Khedmati et al., 2020). Nevertheless, exceptions to this trend exist. For instance, Parmezan et al. (2019) find that SARIMA outperforms specific ML algorithms, particularly those characterized by a larger number of parameters, while (Kumar and Patil, 2015) report that ARIMA delivers superior volatility forecasts over NN for a ten-day horizon.

As indicated by Yarovaya et al. (2021) and Koutmos (2023), the behaviors of participants in virtual currency markets are time-varying. This temporal variation can lead to regime changes in Bitcoin prices and volatilities, which implies that conventional regression-based models focusing on the centre of the return distribution may produce misleading estimates. To overcome such limitations in modeling non-linear dynamics and structural shifts, various regime-switching models have been proposed. For example, Buthelezi (2024) use a Markov-Switching (MS) vector autoregressive model and uncover a strong positive relationship between lagged and current Bitcoin returns. In a related study, Ardia et al. (2019) find that MSGARCH models outperform single-regime GARCH specifications in capturing the volatility of Bitcoin returns. Similarly, Bouri et al. (2022) show that regime-switching factor models enhance the forecasting performance for returns of major cryptocurrencies. Nevertheless, Hotta et al. (2025) highlight the importance of careful specification of regime-switching models, as such specifications are significantly associated with model performance. From a ML perspective, MS-type econometric models for regime changes can be viewed as a specific application of Hidden Markov Models in finance. This interpretation underscores the alignment between advanced econometric methods and the broader class of machine learning models, thereby emphasizing the advantages of adopting ML-based approaches in capturing

complex, latent structures in financial time series. In fact, many studies show that ML techniques generally exhibit superior predictive performance relative to conventional econometric approaches like GARCH-family models (D'Ecclesia and Clementi, 2021; Amirshahi and Lahmiri, 2023; Sørensen, 2023; García-Medina and Aguayo-Moreno, 2024; Ampountolas, 2024; Tjøstheim, 2025). However, Chatterjee et al. (2022) observe that LSTM networks outperform competing models in predicting volatility within the pharmaceutical sector, but not in banking and IT sectors. In contrast, E-GARCH demonstrates superior performance in the banking sector, while GJR-GARCH models yield better forecasts for the IT and pharmaceutical sectors. Similarly, Hossain and Nasser (2008) and Lian et al. (2022) find that GARCH-type models provide more accurate forecasts than NN in certain contexts.

Finally, hybrid models, some incorporating long-memory features, typically exhibit enhanced predictive accuracy relative to standalone models (Yan and Ouyang, 2018; Júnior et al., 2019; Nosratabadi et al., 2020; Li et al., 2020a; Ciaburro and Iannace, 2021). However, studies such as (Dingli and Fournier, 2017) and García-Medina and Aguayo-Moreno (2024) challenge this consensus, suggesting that simpler models may outperform complex hybrid models for specific forecasting tasks.

Although ML methods have generally demonstrated superior predictive performance in financial time series forecasting compared to traditional econometric models, the existing literature reveals notable areas requiring further exploration.

First, despite the promising performance of ML algorithms, their interpretability remains a substantial limitation. Lommers et al. (2021) underscore the necessity of interpretability and explainability for deriving meaningful financial insights. Given the inherent "black-box" nature of ML approaches, characterized by high-dimensional data and numerous parameters, it becomes challenging to dissect their learning processes or to clearly attribute forecasting outcomes to specific model features. Consequently, comparative studies frequently arrive at divergent conclusions regarding which ML model is most effective for financial time series forecasting.

Second, the rapid advancement and proliferation of ML models exacerbate the difficulty of appropriate model selection. Literature surveys and empirical comparisons typically illustrate considerable variability in performance rankings across diverse studies. For instance, while (Ahmed et al., 2010; Fischer et al., 2018) and Derbentsev et al. (2020) consistently highlight MLP as superior, leading to their widespread use in financial prediction tasks, other researchers advocate support vector machines (SVM) as equally effective alternatives (Reddy and Sai, 2018; Parray et al., 2020). Furthermore, recent empirical studies provide mixed outcomes on the performance of advanced deep learning architectures: (Lian, 2024) identify GRU as outperforming other Seq2Seq models, whereas extensive literature supports attention-based transformers and their variants as achieving superior forecasting accuracy (Wu and Chen, 2020; Sridhar and Sanagavarapu, 2021; Tanwar and Kumar, 2022; Penmetsa and Vemula, 2023; Amadeo et al., 2023; Chalkiadakis et al., 2023; Fang, 2023; Gezici and Sefer, 2024; Singh and Bhat, 2024). Nonetheless, Dingli and Fournier (2017) challenge the prevailing assumption that complex deep learning architectures inherently perform better, providing evidence that simpler models can sometimes yield superior predictions.

Third, the characteristics of input data, particularly issues related to stationarity and memory properties, significantly influence the effectiveness of machine learning models. Livieris et al. (2020) emphasize that stationarity, referring to consistent statistical properties such as mean, variance, and autocorrelation structure across time, is crucial for ensuring the accuracy, stability, and predictive reliability of advanced deep learning methods including convolutional neural networks and LSTM networks. Furthermore, memory attributes, notably long memory or persistent dependence within data, present additional complications by challenging the assumptions of machine learning models. Therefore, adequately addressing the capability of machine learning models to handle these specific data characteristics is essential for optimizing model performance and ensuring robust forecasting outcomes in practical applications.

Fourth, due to the inherent complexity and black-box characteristics of advanced models, especially deep learning architectures such as Seq2Seq models, it remains challenging to reliably evaluate their effectiveness in capturing long-memory dependence within time series data. Unlike traditional statistical methods, where significance tests can readily determine the importance and interpretability of individual parameters, complicated machine learning models typically lack such statistical frameworks, making formal tests of parameter significance infeasible. This further complicates our understanding of how effectively these sophisticated models are learning from a long-memory time series.

Finally, existing research rarely explicitly incorporates the systemic long-memory characteristics inherent to financial data, potentially resulting in misleading assessments of ML model performance. Commonly, evaluation relies solely on predictive metrics like MSE, without considering the consistency of predicted series' statistical properties. Moreover, studies explicitly performing statistical consistency checks concerning long-memory attributes remain notably scarce.

Addressing these gaps necessitates rigorous empirical investigation aimed at assessing whether ML models genuinely possess the capacity to capture and replicate long-memory behaviors inherent in financial data. Such analyzes could guide a more informed model specification process, enhancing forecasting accuracy from the perspective of capturing and modeling long-memory phenomena, contributing to innovative model development, power of model generalization and interpretation.

2.4. Summary of the literature review

Generally, numerous studies have demonstrated the potential of ML outperforming statistical methods in financial time series forecasting with a large sample size. Classical technical indicators do not evidently improve model performance, while transaction data such as opening, closing, high, low prices is popular as input for future price prediction. The interpretability of ML models needs further exploration due to the varied performance observed when processing data with different properties, implying that it is difficult to identify a best model for price prediction. Furthermore, although many powerful models exist, there is a lack of an effective method to assess their capability in handling long-memory series, as the statistical properties of the data significantly influence ML performance. Empirical evidence supports that hybrid methods, which account for data characteristics such as long

memory and stationarity, can outperform single models in terms of forecast accuracy. This raises a pertinent question regarding the ability of ML to process long-memory data, which is thoroughly examined in this paper. Based on the literature reviewed above, we consider prediction as a regression problem, mainly focussing on popular ML models including SVR, RF, MLP, RT, GP, GKR, Linear models, BiLSTM, GRU, LSTM and the attention mechanism.

3. Methodology

3.1. Test designs

The empirical literature provides compelling evidence that asset prices and returns in cryptocurrency markets, notably BTC, exhibit pronounced time-varying long-memory dynamics and heavy-tailed distributions. These complex statistical features challenge the assumption of $i.i.d$ returns, frequently invoked in traditional financial modeling. Despite this well-documented phenomenon, a considerable segment of existing ML applications for return prediction overlooks the structural implications of long memory, potentially weakening model robustness and generalization capacity in dynamic financial environments.

To address this methodological gap, we propose two tests for ML models. The Type 1 test comprises a two-stage evaluation procedure, beginning with simulations in a controlled environment using synthetic data, followed by assessments under real financial conditions to validate the models' capacity to capture long memory in the context of regime switching. Inspired by Mirzaei et al. (2014), we design a generative forecasting task wherein trained models use historical data to generate future series. The degree of long memory in the generated series is then quantified using the FELW estimator for univariate series and FCELW for bivariate dependence. Specifically, in the first stage, following Luo et al. (2021), we generate synthetic datasets from a standard normal distribution to evaluate the baseline learning capacity of various ML models. Only models that demonstrate acceptable performance on synthetic series after this preliminary evaluation are retained for further analysis. This hierarchical design ensures that only empirically validated models, robust to synthetic complexity, are exposed to real-world financial series, thereby filtering candidates on the basis of statistical consistency.

In the second stage, we scrutinize the capacity of these selected models to internalize and replicate long-memory characteristics intrinsic to cryptocurrency markets. By comparing the estimated fractional differencing parameter d with that observed in real BTC returns, we assess whether the ML models effectively capture long-range dependence.

For the second-type test, we benchmark ML-based estimations of long memory against established statistical estimators, including recent innovations in fractional modeling (Ledesma-Orozco et al., 2011; Zheng et al., 2021; Liu et al., 2022; Mukherjee et al., 2023; Boros et al., 2024). If ML approximators exhibit comparable estimation accuracy to FELW or FCELW, they are deemed capable of mimicking long-memory processes, offering novel tools for assessing market inefficiency. This approach not only enhances the empirical validity of our models but also contributes a novel evaluative criterion for the efficiency of cryptocurrency markets from a machine learning perspective.

Approaches for memory parameter d estimation

Univariate memory

To estimate the long-memory parameter d in fractionally integrated processes, we employ the feasible exact local Whittle (FELW) estimator, as developed by Shimotsu (2010). The FELW estimator constitutes an extension of the exact local Whittle (ELW) method introduced by Shimotsu and Phillips (2005), specifically adapted to accommodate time series with unknown deterministic components, such as an unknown mean and polynomial time trends. We consider a fractionally integrated process X_t characterized by:

$$(1-L)^d(X_t-\mu_0-\beta_1 t-\cdots-\beta_k t^k)=u_t,\quad t=1,\ldots,n, \tag{3}$$

where u_t is a stationary process with zero mean and spectral density $f_u(\lambda)\sim G_0$ as $\lambda\to 0$.

The method relies on the modified objective function:

$$R_F(d)=\log\widehat{G}_F(d)-2d\cdot\frac{1}{m}\sum_{j=1}^{m}\log\lambda_j, \tag{4}$$

where:

$$\widehat{G}_F(d)=\frac{1}{m}\sum_{j=1}^{m}I_{\Delta^d(X_t-\widehat{\mu}(d))}(\lambda_j), \tag{5}$$

and $I_{\Delta^d(X_t-\widehat{\mu}(d))}(\lambda_j)$ is the periodogram of the fractionally differenced, demeaned (and detrended if necessary) data evaluated at the Fourier frequencies $\lambda_j=2\pi j/n$.

The estimator $\widehat{d}_{FELW}$ is defined as:

$$\widehat{d}_{FELW}=\arg\min_{d\in[d_1,d_2]}R_F(d). \tag{6}$$

The estimation strategy consists of a two-step procedure. In the first stage, we obtain an initial consistent estimate of d through a tapered local Whittle estimator. This preliminary estimator ensures robustness against possible nonstationarity and deterministic

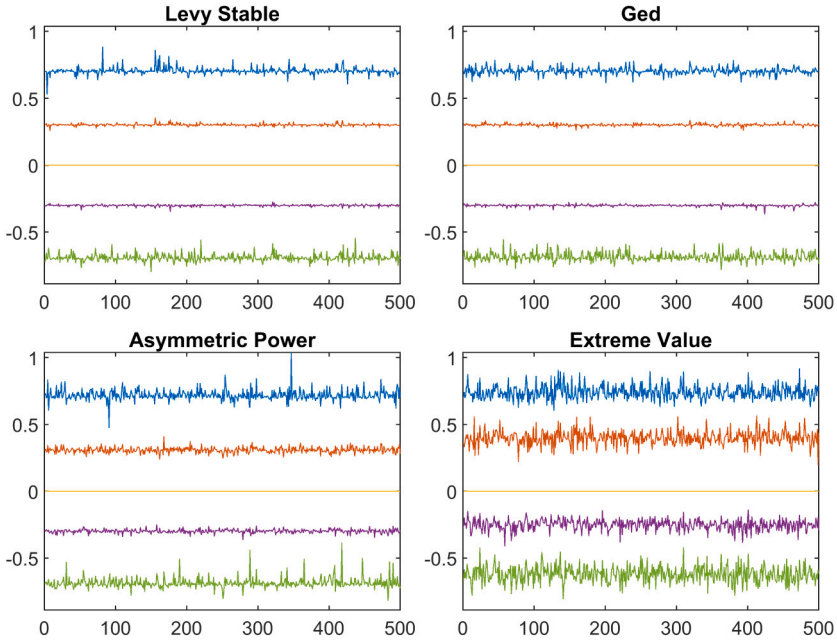


Fig. 7. Robustness tests for FELW across distribution (a).
Note: Fig. 7 illustrates the performance of FELW estimators to estimate series with the value of $-0.7, -0.3, 0, 0.3, 0.7$ across different distributions.

trends in the data. Subsequently, in the second stage, the initial estimate informs the construction of a modified ELW objective function. This objective function incorporates a feasible adjustment for the unknown mean and polynomial trends by applying an appropriate demeaning and detrending process. The FELW estimator is then derived as the minimizer of the modified objective function over the admissible parameter space of d.

Under general regularity conditions, the FELW estimator is shown to be consistent and asymptotically normal for $d \in \left(-\frac{1}{2}, 2\right)$, and up to $d \in \left(-\frac{1}{2}, \frac{7}{4}\right)$ when the data exhibit polynomial trends. Notably, the FELW estimator maintains the desirable properties of the ELW approach, including asymptotic efficiency and distributional robustness, while also effectively addressing the potential complications arising from deterministic components in the observed time series.

1. **First Step:** Obtain an initial consistent estimator $\widehat{d}_T$ using a tapered local Whittle estimator to mitigate the impact of deterministic trends.
2. **Second Step:** Substitute $\widehat{d}_T$ into the weight function $w(d)$, which combines the sample mean $\bar{X}$ and the first observation X_1, to construct an estimator of the mean (or initial condition):

$$\widehat{\mu}(d) = w(d) \cdot \bar{X} + (1 - w(d)) \cdot X_1, \tag{7}$$

where $w(d)$ satisfies:

$$w(d) = \begin{cases} 1, & d \leq \frac{1}{2} \\ 0, & d \geq \frac{3}{4}. \end{cases}$$

3. Optimize $R_F(d)$ based on $\widehat{\mu}(d)$ to obtain $\widehat{d}_{FELW}$.

The FELW estimators exhibit remarkable robustness across a broad range of distributional settings, including heavy-tailed (e.g., Student-t, Generalized Pareto), skewed (e.g., Skewed GED, Asymmetric Power), and leptokurtic (e.g., Generalized Hyperbolic) cases. As illustrated in Figs. 7 to 9, the estimated sequences remain well-contained and centred around their respective differencing levels, with minimal distortion from the underlying distributional features. This suggests that the FELW estimator maintains its effectiveness regardless of the tail behavior or skewness present in the innovations.

Systemic memory

Systemic memory is estimated by the Fractionally Cointegrated Exact local Whittle (FCELW) estimator by Shimotsu (2012), who adopts a two-step semiparametric procedure to estimate the memory parameter d, which captures the degree of fractional integration in fractionally cointegrated systems. The approach is designed to accommodate both stationary ($d < 0.5$) and non-stationary ($d \geq 0.5$) processes, ensuring robust estimation across different regimes.

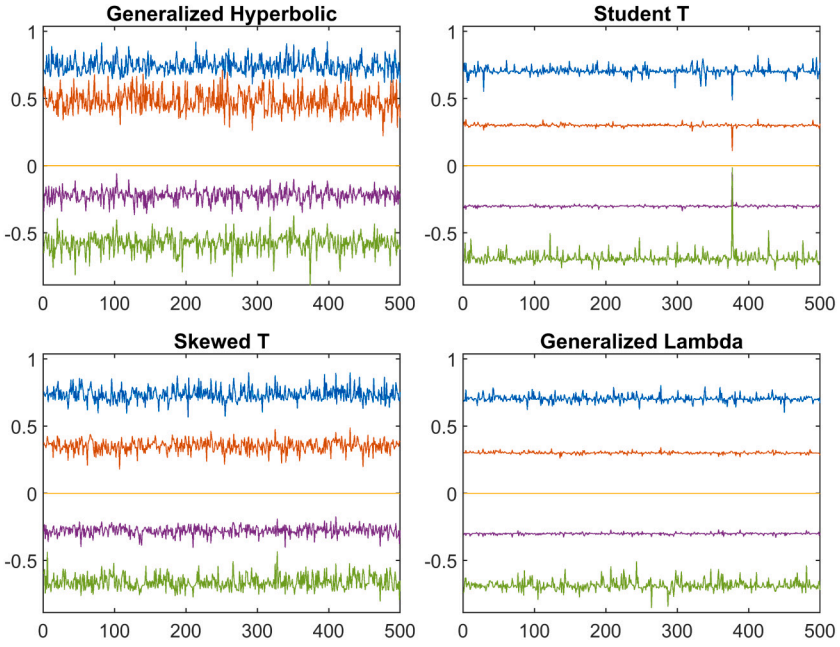


Fig. 8. Robustness tests for FELW across distribution (b).
Note: Fig. 8 illustrates the performance of FELW estimators to estimate series with the value of $-0.7, -0.3, 0, 0.3, 0.7$ across different distributions.

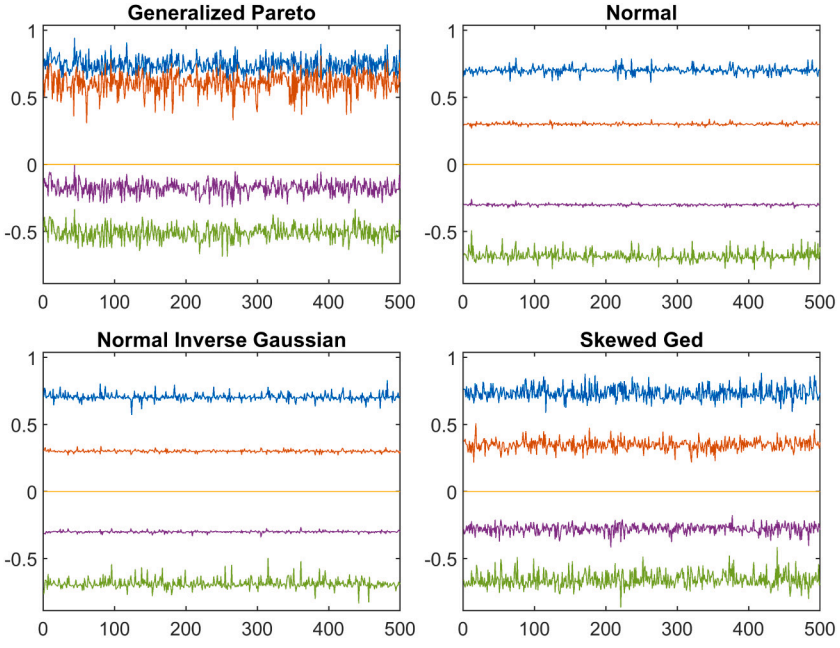


Fig. 9. Robustness tests for FELW across distribution (c).
Note: Fig. 9 illustrates the performance of FELW estimators to estimate series with the value of $-0.7, -0.3, 0, 0.3, 0.7$ across different distributions.

Consider the following bivariate system:

$$\begin{cases} (1-L)^{d_1}(y_t - \beta x_t) = u_{1t}, & t = 1, 2, \ldots \\ (1-L)^{d_2} x_t = u_{2t}, & t = 1, 2, \ldots \end{cases} \tag{8}$$

where $u_t = (u_{1t}, u_{2t})'$ is a stationary zero-mean vector process with spectral density matrix $f_u(\lambda)$. The parameters d_1 and d_2 represent the memory parameters of the equilibrium error and the common stochastic trend, respectively. It is assumed that $-\frac{1}{2} < d_1 < d_2 < \overline{d} < \infty$.

The first-step estimator employs a tapered local Whittle method adjusted by trimming and tapering to ensure consistency over the range of d_1 and d_2. The estimator minimizes the following objective function:

$$R(\vartheta) = \log\det \widehat{\Omega}_T(\vartheta) - \frac{2(d_1 + d_2)}{p(1-\kappa)m} \sum_{j(p,\kappa)}^{m} \log \lambda_j, \tag{9}$$

where $\vartheta = (\beta, d_1, d_2)'$, and $\widehat{\Omega}_T(\vartheta)$ is defined as

$$\widehat{\Omega}_T(\vartheta) = \frac{p}{(1-\kappa)m} \sum_{j(p,\kappa)}^{m} \mathrm{Re}\left[\Psi(\lambda_j; d) B I_{Tz}(\lambda_j) B' \overline{\Psi}(\lambda_j; d)\right], \tag{10}$$

with

$$\Psi(\lambda_j; d) = \mathrm{diag}\left(\lambda_j^{d_1}, \lambda_j^{d_2} e^{-i(\pi-\lambda_j)(d_2-d_1)/2}\right), \tag{11}$$

and

$$B = \begin{bmatrix} 1 & -\beta \\ 0 & 1 \end{bmatrix}, \quad I_{Tz}(\lambda_j) = w_{Tz}(\lambda_j) w_{Tz}(\lambda_j)^*. \tag{12}$$

Here, $w_{Tz}(\lambda_j)$ denotes the tapered discrete Fourier transform (DFT) of the observed vector $z_t = (y_t, x_t)'$, and $\lambda_j = 2\pi j/n$.

The second step refines the estimates by minimizing the exact local Whittle objective function:

$$R^*(\vartheta) = \log\det \widetilde{\Omega}^*(\vartheta) - \frac{2(d_1 + d_2)}{m} \sum_{j=1}^{m} \log \lambda_j, \tag{13}$$

where

$$\widetilde{\Omega}^*(\vartheta) = \frac{1}{m} \sum_{j=1}^{m} \mathrm{Re}\left[I_{\Delta^d z}(\lambda_j; \beta)\right], \tag{14}$$

and

$$I_{\Delta^d z}(\lambda_j; \beta) = w_{\Delta^d z}(\lambda_j; \beta) w_{\Delta^d z}(\lambda_j; \beta)^*, \tag{15}$$

with

$$w_{\Delta^d z}(\lambda_j; \beta) = \begin{pmatrix} w_{\Delta^{d_1}}(y_t - \beta x_t)(\lambda_j) \\ w_{\Delta^{d_2}} x_t(\lambda_j) \end{pmatrix}. \tag{16}$$

The exact local Whittle estimator $\vartheta^* = (\beta^*, d_1^*, d_2^*)'$ obtained from (13) satisfies the following asymptotic distribution:

$$m^{1/2} \Delta_n^* (\vartheta^* - \vartheta_0) \xrightarrow{d} N(0, \Xi^{-1}), \tag{17}$$

where

$$\Delta_n^* = \mathrm{diag}\left(\lambda_m^{-\nu_0}, 1, 1\right), \quad \nu_0 = d_2 - d_1, \tag{18}$$

provided $\nu_0 \in (0, 1/2)$.

This two-step estimation procedure, combining the tapered local Whittle and exact local Whittle approaches, ensures consistent and asymptotically normal estimates of the memory parameters d_1 and d_2, as well as the cointegrating coefficient β, under both stationary and non-stationary conditions.

3.2. Open-loop and closed-loop forecasting

In time series forecasting, particularly when employing RNN, open-loop forecasting and closed-loop forecasting are two common strategies used to predict future time steps. These approaches differ in how they utilize the data for making predictions and in their application contexts.

Open-loop forecasting

Open loop forecasting involves predicting the next time step in a sequence using only the available historical data. This approach requires the true values of the data source at each step to make subsequent predictions. Specifically, to forecast the value at time step $t+1$, the model first uses the true values from time steps 1 to t. After making the prediction for $t+1$, the actual observed value at $t+1$ is recorded and used as input to predict the next step $t+2$. This method is iterative, and the model continuously receives the actual data as feedback. This strategy is particularly effective when the true values of the sequence are available during the prediction process. This ensures that each step of the forecast is based on accurate and up-to-date information, minimizing the risk of error accumulation over time.

Closed-loop forecasting

Closed loop forecasting, in contrast, is a technique in which subsequent time steps are predicted by feeding the model's own previous predictions as input, rather than the true values. This approach is essential when true values are not available during the prediction phase, which is often the case in real-world forecasting scenarios. To illustrate, consider predicting the values for the time steps t through $t+k$. In this scenario, after making a prediction for the time step t, this predicted value is then used as input to predict the next time step, $t+1$, and so on, until all future desired time steps are predicted. This method is useful for generating forecasts for multiple time steps when only historical data up to time step $t-1$ are available. This strategy can lead to a propagation of errors, especially in long-term forecasts, since any inaccuracy in one prediction is carried forward to subsequent predictions. However, it is often the only feasible approach when future true values are unavailable and is valuable in scenarios requiring extended forecasts.

3.3. Gaussian process regression

According to Rasmussen (2003), Gaussian Process Regression (GPR) is a nonparametric Bayesian approach to regression that has gained popularity in various fields due to its flexibility and ability to quantify uncertainty. The essence of GPR is to place a Gaussian process prior over the space of functions, and then use observed data to update this prior and obtain a posterior distribution over functions. This approach allows for a principled way to incorporate prior knowledge and handle noise in the data.

GP is a collection of random variables, any finite number of which have a joint Gaussian distribution. GP is fully specified by its mean function $m(\mathbf{x})$, and its covariance function $k(\mathbf{x}, \mathbf{x}')$. Formally, if $f(\mathbf{x})$ is a function defined by a GP, we write:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \tag{19}$$

where $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ and $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$.

In the context of regression, we assume that our observations $\mathbf{y}$ are generated from an unknown function $f(\mathbf{x})$ with added Gaussian noise. Specifically,

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_n^2), \tag{20}$$

where ϵ_i represents independent Gaussian noise with variance σ_n^2.

Given a set of training data $\{\mathbf{X}, \mathbf{y}\}$, where $\mathbf{X}$ is the matrix of input vectors and $\mathbf{y}$ is the vector of observed outputs, the goal of GPR is to predict the value of $f(\mathbf{x}_*)$ for a new input $\mathbf{x}_*$. The joint distribution of the observed outputs $\mathbf{y}$ and the function value $f(\mathbf{x}_*)$ at a test point $\mathbf{x}_*$ is given by:

$$\begin{pmatrix} \mathbf{y} \\ f(\mathbf{x}_*) \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mathbf{m}(\mathbf{X}) \\ m(\mathbf{x}_*) \end{pmatrix}, \begin{pmatrix} K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I & K(\mathbf{X}, \mathbf{x}_*) \\ K(\mathbf{x}_*, \mathbf{X}) & k(\mathbf{x}_*, \mathbf{x}_*) \end{pmatrix} \right), \tag{21}$$

where $K(\mathbf{X}, \mathbf{X})$ is the covariance matrix evaluated in the training inputs, $K(\mathbf{X}, \mathbf{x}_*)$ is the vector of covariances between the training inputs and the test input, and $k(\mathbf{x}_*, \mathbf{x}_*)$ is the covariance between the test input and itself.

To make predictions, we condition the joint Gaussian distribution on the observed data. The predictive distribution for $f(\mathbf{x}_*)$ given the training data is also Gaussian, with mean and covariance given by:

$$\mathbb{E}[f(\mathbf{x}_*) \mid \mathbf{X}, \mathbf{y}, \mathbf{x}_*] = m(\mathbf{x}_*) + K(\mathbf{x}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I]^{-1}(\mathbf{y} - \mathbf{m}(\mathbf{X})), \tag{22}$$

$$\mathrm{Cov}(f(\mathbf{x}_*) \mid \mathbf{X}, \mathbf{y}, \mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I]^{-1} K(\mathbf{X}, \mathbf{x}_*). \tag{23}$$

These equations demonstrate how the posterior mean provides the best estimate of the function value at a new input, and the posterior covariance quantifies the uncertainty associated with this estimate. The GPR can be visualized as Fig. 10.

3.4. Support vector regression

SVR by Drucker et al. (1996) is a supervised learning method used for regression tasks. It extends the principles of SVM by Cortes and Vapnik (1995), a popular technique for classification, to handle regression problems. The primary goal of SVR is to find a function that approximates the relationship between input features and continuous output values with a high degree of precision while maintaining robustness against overfitting.

SVR operates by mapping the input data into a higher-dimensional feature space using a kernel function. This transformation allows the model to capture non-linear relationships between the input and output variables. The central idea in SVR is to find a function $f(x)$ that deviates from the actual target values y by at most ϵ for each training point, while simultaneously ensuring that the complexity of the model is minimized.

Mathematically, given a training dataset $\{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ represents the input features and $y_i \in \mathbb{R}$ represents the corresponding target values, the SVR aims to find a function

$$f(x) = w \cdot \phi(x) + b, \tag{24}$$

where $\phi(x)$ is the feature transformation, w is the weight vector, and b is the bias term.

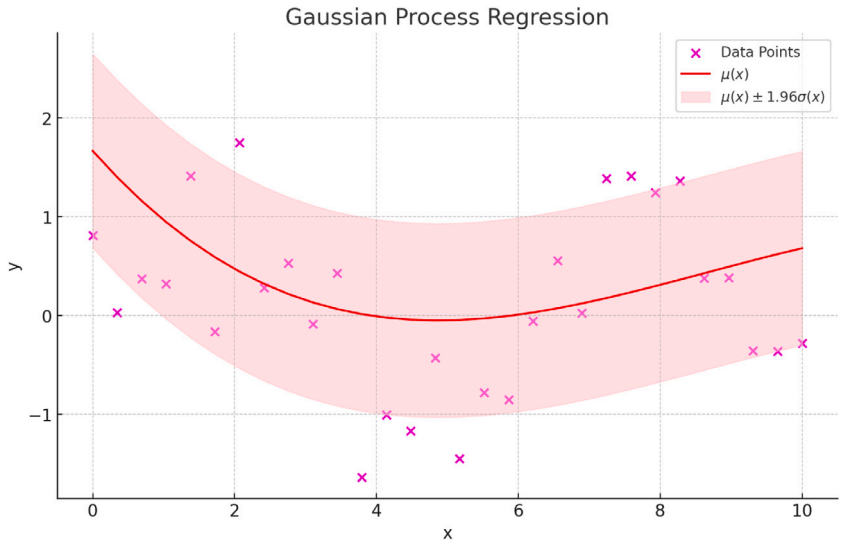


Fig. 10. Visualization of Gaussian Process Regression.
Note: Fig. 10 illustrates the results of GPR. The magenta crosses represent observed data points. The red solid line denotes the predictive mean $\mu(x)$, while the shaded pink area indicates the 95% confidence interval, given by $\mu(x) \pm 1.96\sigma(x)$. The width of the confidence band reflects the uncertainty of the prediction, which increases in regions with fewer data points. This property highlights the ability of GPR to provide both point predictions and measures of predictive uncertainty. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The optimization problem in SVR can be formulated as follows:

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*), \tag{25}$$

subject to:

$$\begin{cases} y_i - (w \cdot \phi(x_i) + b) \le \epsilon + \xi_i \\ (w \cdot \phi(x_i) + b) - y_i \le \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \quad \forall i. \end{cases} \tag{26}$$

In this formulation, ϵ is a user-defined parameter that specifies the maximum acceptable deviation from the true target values y_i, ξ_i and ξ_i^* are slack variables introduced to allow for some errors in the model, and C is a regularization parameter that controls the trade-off between the flatness of the regression function and the amount to which deviations larger than ϵ are tolerated.

The dual form of the SVR optimization problem, which is often more convenient for computation, is given by:

$$\max_{\alpha,\alpha^*} \left\{ -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(x_i, x_j) - \epsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \right\}, \tag{27}$$

subject to:

$$\begin{cases} \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \\ 0 \le \alpha_i, \alpha_i^* \le C \quad \forall i. \end{cases} \tag{28}$$

Here, α_i and α_i^* are the Lagrange multipliers and $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ is the kernel function. The kernel function allows SVR to operate in the transformed feature space without explicitly computing the transformation $\phi(x)$, enabling the handling of high-dimensional or even infinite-dimensional spaces.

After solving the dual problem, the regression function can be expressed as:

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(x_i, x) + b. \tag{29}$$

The visualization of SVR can be shown in Fig. 11.

SVR is advantageous in scenarios where the relationship between the input variables and the output is complex and non-linear. By selecting an appropriate kernel function, such as linear, polynomial, or RBF, SVR can capture intricate patterns in the data. Moreover, the regularization parameter C and the precision parameter ϵ provide flexibility in controlling the complexity of the model and the tolerance to deviations, respectively, thereby offering a robust mechanism for regression analysis in various applications.

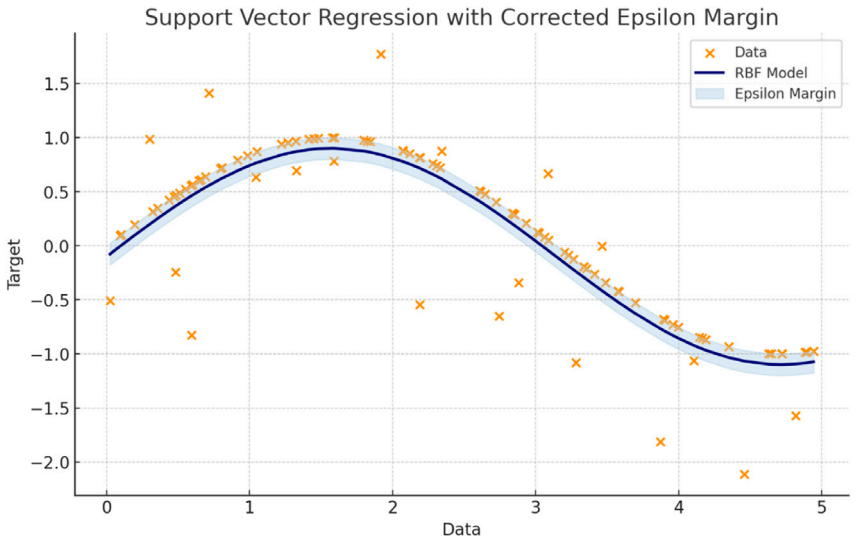


Fig. 11. Visualization of support vector regression.
Note: Fig. 11 illustrates the results of SVR with a corrected epsilon margin. The orange crosses denote observed data points. The dark blue line represents the fitted SVR model with a radial basis function (RBF) kernel, while the light blue shaded area shows the epsilon margin around the fitted curve. This margin highlights the region within which prediction errors are not penalized by the SVR loss function, reflecting the robustness of the model to noise in the data. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

To implement the kernel trick within SVR, various kernel functions can be utilized to transform the input space into a higher-dimensional feature space. These kernel functions include the linear kernel, defined as

$$K(x_i, x_j) = x_i^T x_j, \tag{30}$$

which does not transform the data and is equivalent to the standard dot product in the input space. Another commonly used kernel is the polynomial kernel, represented by

$$K(x_i, x_j) = (x_i^T x_j + c)^d, \tag{31}$$

where c is a constant trading off the influence of higher-order terms and d is the degree of the polynomial, enabling non-linear modeling by mapping data to higher dimensions. The RBF kernel, defined as

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \tag{32}$$

where σ is a free parameter, is particularly powerful for non-linear data as it maps the data into an infinite-dimensional space. The sigmoid kernel,

$$K(x_i, x_j) = \tanh(\kappa x_i^T x_j + c), \tag{33}$$

with parameters κ and c, is related to the neural network's activation function. Other kernels include the exponential kernel,

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|}{2\sigma^2}\right), \tag{34}$$

and the Laplacian kernel,

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|_1}{\sigma}\right), \tag{35}$$

which are variants of the RBF kernel using different distance measures. These kernel functions allow SVR to handle various data types by implicitly mapping them to higher-dimensional spaces, facilitating linear separation.

An example of the kernel function processing linearly inseparable problems can be visualized in Figs. 12 and 13.

3.5. Linear regression

Linear regression is a fundamental statistical method used to model the relationship between a dependent variable and one or more independent variables. The simplest form, simple linear regression, involves one independent variable. The model is defined

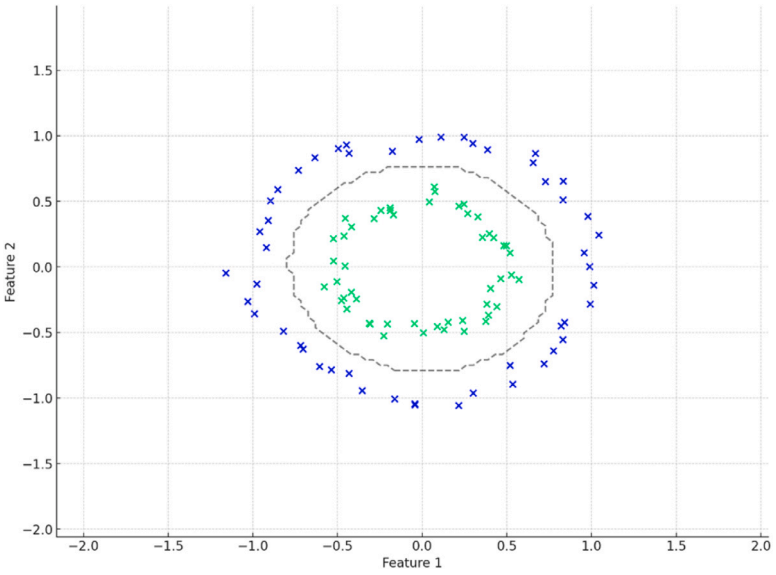


Fig. 12. Linearly inseparable data.
Note: Fig. 12 shows an example of linearly inseparable data in two dimensions. The green and blue crosses represent two different classes that cannot be separated by a linear decision boundary. The dashed curves indicate possible non-linear boundaries that could separate the classes, illustrating the need for non-linear classification methods such as kernel-based support vector machines. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

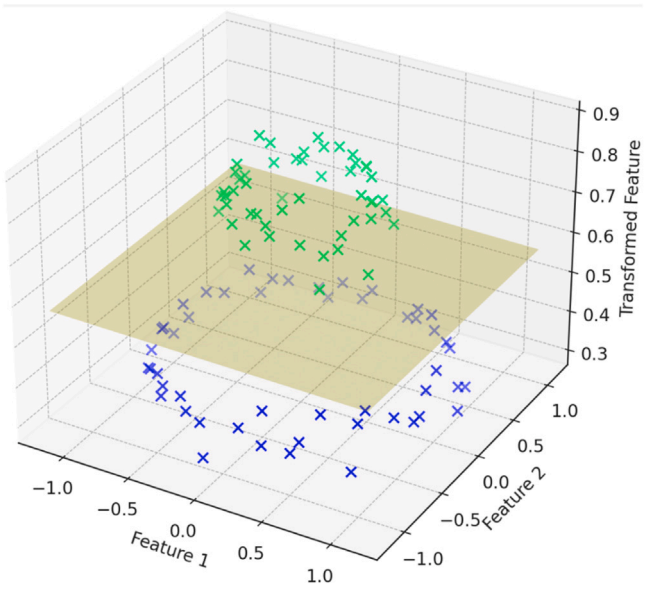


Fig. 13. Example for mapping data to higher dimension by the kernel function.
Note: Fig. 13 demonstrates how linearly inseparable data in two dimensions can become linearly separable when mapped to a higher-dimensional feature space using a kernel function. The green and blue crosses correspond to two different classes, and the additional dimension enables the separation of the classes by a linear decision surface, as illustrated by the plane. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

as follows.

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \tag{36}$$

where y_i is the dependent variable. x_i is the independent variable. β_0 is the intercept. β_1 is the slope coefficient. ϵ_i is the error term.

For multiple linear regression, involving p independent variables, the model is expressed as:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i. \tag{37}$$

The objective is to estimate the coefficients $\beta_0, \beta_1, \ldots, \beta_p$ by minimizing the sum of squared residuals:

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2. \tag{38}$$

Linear regression is suitable for modeling the relationship between variables without regularization. However, it may suffer from overfitting if the number of predictors is large relative to the number of observations.

Lasso regression

Lasso regression proposed by Tibshirani (1996), or the least absolute shrinkage and selection operator, extends linear regression by adding a penalty equivalent to the absolute value of the magnitude of the coefficients. This form of regularization not only helps in preventing overfitting, but also performs variable selection by shrinking some coefficients to zero. The objective lasso regression function is:

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}, \tag{39}$$

where λ is the regularization parameter that controls the degree of shrinkage. A larger λ results in more coefficients shrinking towards zero.

Ridge regression

Ridge regression, also known as Tikhonov regularization by Hoerl and Kennard (1970), adds a penalty equivalent to the square of the magnitude of the coefficients to the linear regression objective. This regularization technique is particularly useful for dealing with multicollinearity, where independent variables are highly correlated. The objective function of the ridge regression is the following:

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}. \tag{40}$$

Similarly to lasso regression, λ is the regularization parameter. However, unlike lasso, ridge regression does not perform variable selection but rather shrinks all coefficients towards zero proportionally.

Lasso regression is useful when we expect that many predictors have no effect on the response variable, effectively performing feature selection. Ridge regression is beneficial when dealing with multicollinearity, ensuring that coefficient estimates are more stable and less sensitive to changes in the model. Both lasso and ridge regression can be combined in a method known as Elastic Net, which incorporates both $L1$ and $L2$ penalties, providing a balance between the two techniques.

3.6. Regressive tree models and random forest regression

Regressive Tree Models

Regressive tree models, also known as regression trees, are a type of decision tree that is used to predict continuous results. These models belong to a broader category of ML techniques, called decision trees. The fundamental concept of a regression tree is to partition the data space into a set of rectangles and fit a simple model (typically a constant) in each one. The partitions are created using recursive binary splits of the feature space.

The general form of a regression tree model can be described mathematically. Let $D = \{(x_i, y_i)\}_{i=1}^{n}$ represent the dataset, where $x_i \in \mathbb{R}^p$ are the input characteristics and $y_i \in \mathbb{R}$ is the target variable. The regression tree partitions the feature space $\mathbb{R}^p$ into M disjoint regions R_m, $m = 1, 2, \ldots, M$, and fits a constant value c_m in each region. The prediction $\hat{y}$ for an input x is given by:

$$\hat{y} = \sum_{m=1}^{M} c_m I(x \in R_m), \tag{41}$$

where $I(\cdot)$ is an indicator function that equals 1 if the condition is true and 0 otherwise. The constants c_m are typically the mean value of the target variable y in the region R_m. RT models can be visualized as Fig. 14.

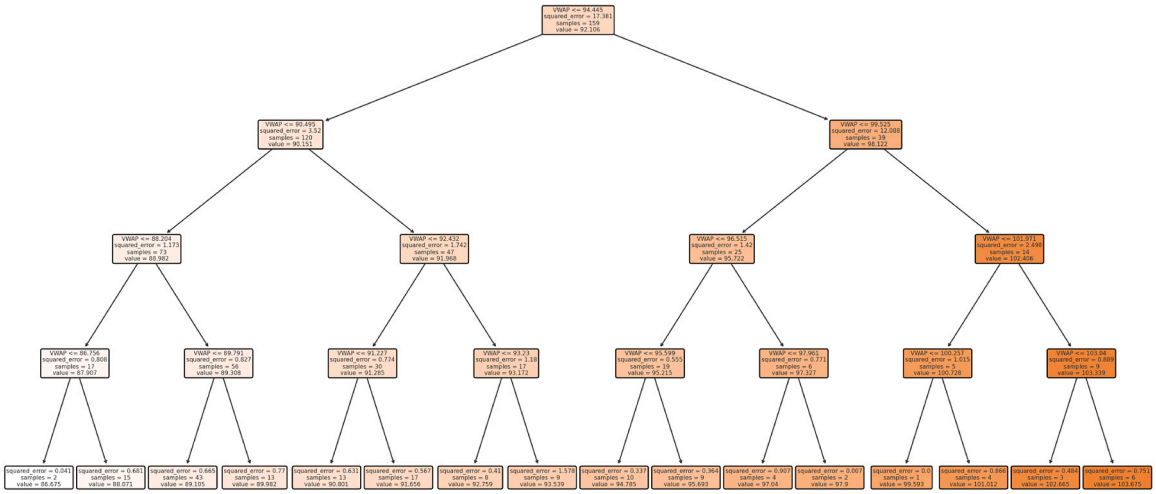


Fig. 14. Visualization of regression tree.
Note: Fig. 14 visualizes a regression tree model. The tree structure represents a sequence of binary splits on the input variables, with each internal node indicating a decision rule and each leaf node representing a predicted value. This hierarchical structure enables the model to capture non-linear relationships between features and the target variable.

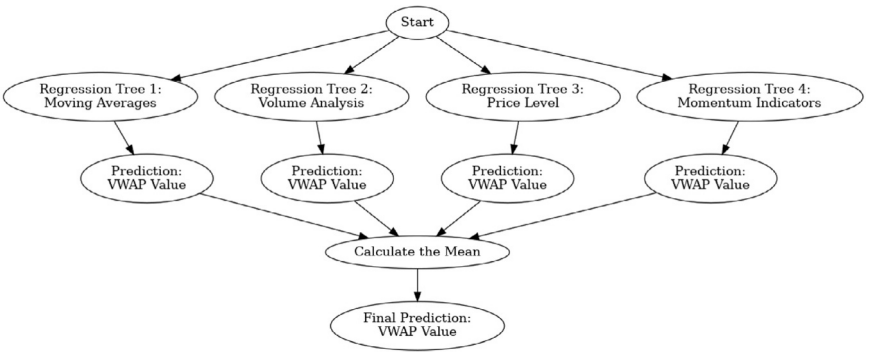


Fig. 15. Visualization of random forest.
Note: Fig. 15 illustrates the structure of a random forest model for regression. Each regression tree independently predicts the VWAP value based on different feature sets (such as moving averages, volume analysis, price level, and momentum indicators). The final prediction is obtained by averaging the outputs of all trees, which enhances robustness and reduces overfitting compared to a single decision tree.

Random forest regression

Random forest regression proposed by Breiman (2001) is an ensemble learning method that improves the predictive accuracy and robustness of single regression trees by averaging multiple trees. Each tree in a random forest is trained on a bootstrapped sample of the original dataset, and at each split in the tree, a random subset of features is considered. This randomization helps reduce the correlation between trees, leading to a more stable and accurate model.

Mathematically, the prediction $\hat{y}$ for an input x using a random forest with trees B can be expressed as follows:

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} \hat{y}^{(b)}(x), \tag{42}$$

where $\hat{y}^{(b)}(x)$ is the prediction of the b -th tree in the forest. Each tree $\hat{y}^{(b)}$ is constructed using a different bootstrapped sample of the data and considers a random subset of features at each split.

The random forest algorithm is robust to overfitting due to the law of large numbers, as the averaging process reduces the variance of the model. This method is widely used in various fields for its high accuracy and interpretability. RF models can be visualized as Fig. 15.

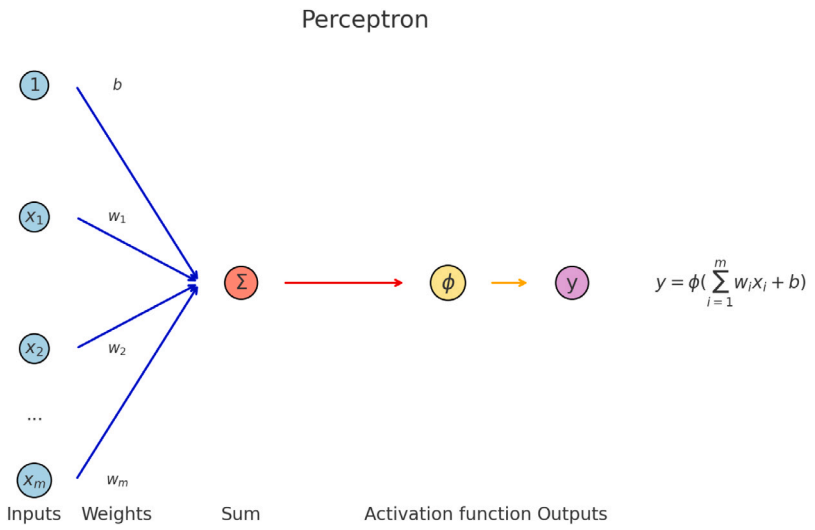


Fig. 16. Visualization of perceptron.
Note: Fig. 16 illustrates the structure of a single-layer perceptron. The inputs (including a bias term) are linearly combined using associated weights and summed. The result is then passed through an activation function $\phi(\cdot)$ to produce the output y. The equation on the right formalizes this process: $y = \phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$.

3.7. Perceptron and multi-layer perceptron

The perceptron is a fundamental building block in the domain of neural networks and serves as the simplest form of a neural network. It was introduced by Rosenblatt (1958) as a linear classifier for binary classification tasks. We visualize it as Fig. 16.

The mathematical formulation of a perceptron can be described as follows.

$$y = \phi(\mathbf{w} \cdot \mathbf{x} + b), \tag{43}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is the weight vector, b is the bias term, ϕ is the activation function, often a step function.

The perceptron operates by computing a weighted sum of its inputs, adding a bias, and then applying an activation function to produce the output. The learning process involves adjusting the weights and bias to minimize the classification error on a given training dataset.

The limitations of the perceptron, particularly its inability to solve linearly non-separable problems, led to the development of more sophisticated models, such as the MLP. The MLP by Rosenblatt (1961) is a class of feedforward artificial neural networks that consists of multiple layers of nodes, typically an input layer, one or more hidden layers, and an output layer. Each node (neurone) in a layer is connected to every node in the subsequent layer through weighted connections.

The mathematical formulation for an MLP can be described as follows.

$$y = \phi(\mathbf{W}^{(L)} \cdot \phi(\mathbf{W}^{(L-1)} \cdot \phi(\cdots \phi(\mathbf{W}^{(1)} \cdot \mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \cdots) + \mathbf{b}^{(L-1)}) + \mathbf{b}^{(L)}, \tag{44}$$

where L denotes the number of layers, $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ are the weight matrix and bias vector of the ith layer, respectively, ϕ is the activation function, which can vary between layers (e.g. ReLU, sigmoid).

The MLP is capable of approximating any continuous function given sufficient hidden units, which is formalized by the Universal Approximation Theorem (Hornik et al., 1989). This capability arises from the non-linear activation functions used in hidden layers, allowing the network to capture complex patterns and interactions within the data. The visualization implemented by Smilkov and Carter (2023) is presented in Fig. 17.

3.8. Long short-term memory

LSTM networks by Hochreiter and Schmidhuber (1997) are a type of RNN architecture designed to overcome the limitations of traditional RNN, particularly their difficulty learning long-term dependence due to issues such as vanishing gradients. LSTM networks achieve this by incorporating memory cells that can maintain information over long time periods.

The core component of an LSTM is the memory cell, which is controlled by three types of gates: the input gate, the forget gate, and the output gate. These gates regulate the flow of information into, out of, and within the memory cell, respectively.

The equations governing the operations of LSTM cells are as follows:

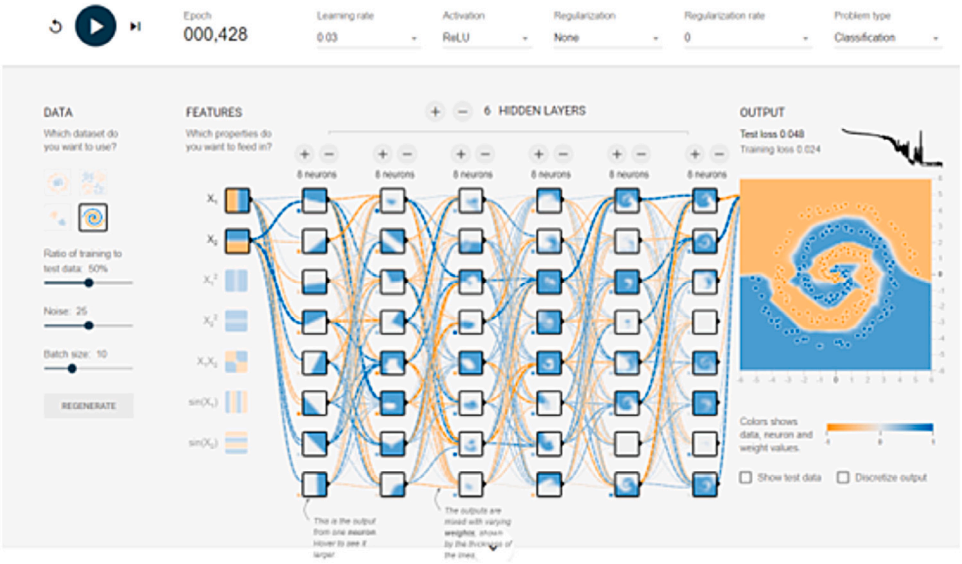


Fig. 17. Visualization of MLP.
Note: Fig. 17 visualizes a MLP for classification. The network consists of an input layer, six hidden layers each with eight neurons, and an output layer. Blue and orange lines indicate the sign and strength of weights. The right panel shows the decision boundary learned by the model on a spiral dataset, with the colour representing class predictions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Forget gate

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f). \tag{45}$$

The forget gate decides what information to discard from the cell state. Here, f_t is the forget gate vector, W_f is the weight matrix, h_{t-1} is the hidden state of the previous time step, x_t is the input at the current time step, b_f is the bias and σ is the activation function of the sigmoid.

Input gate

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{46}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C). \tag{47}$$

The input gate determines what new information to store in the cell state. i_t is the input gate vector, W_i and W_C are weight matrices, $\tilde{C}_t$ is the candidate cell state, and b_i and b_C are biases.

Cell state update

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t. \tag{48}$$

The cell state C_t is updated by combining the previous cell state C_{t-1}, modulated by the forget gate, and the new candidate cell state, modulated by the input gate.

Output gate

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{49}$$

$$h_t = o_t \cdot \tanh(C_t). \tag{50}$$

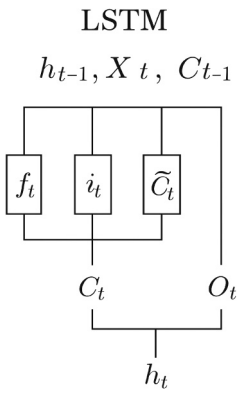


Fig. 18. Visualization of LSTM.
Note: Fig. 18 illustrates the architecture of a LSTM cell. The cell receives the previous hidden state h_{t-1}, the current input X_t, and the previous cell state C_{t-1}. The forget gate f_t, input gate i_t, and candidate cell state $\tilde{C}_t$ regulate the update of the cell state C_t and the output gate O_t determines the hidden state h_t. This structure enables the LSTM to capture long-term dependence in sequential data.

The output gate controls the output from the cell. o_t is the output gate vector, W_o is the weight matrix, b_o is the bias, and h_t is the hidden state at the current time step.

The architecture of LSTM allows it to capture long-term dependence more effectively than traditional RNN, making it highly suitable for tasks that involve sequential data, such as time series forecasting, natural language processing, and speech recognition. The data flow can be visualized as Fig. 18.

3.9. Bidirectional long short-term memory

BiLSTM networks by Graves and Schmidhuber (2005) extend the traditional LSTM architecture by incorporating two separate LSTM layers that process the input sequence in both forward and backward directions. This bidirectional approach allows BiLSTMs to capture contextual information from both past and future states, making them particularly effective for tasks where understanding the context from both directions is crucial, such as natural language processing and speech recognition.

In a BiLSTM, the forward LSTM processes the sequence from the beginning to the end, while the backward LSTM processes the sequence from the end to the beginning. The hidden states from both directions are concatenated to form the final output.

The equations for a BiLSTM can be described as follows:

Forward LSTM

$$\overrightarrow{h_t} = \text{LSTM}_{\text{forward}}(x_t, \overrightarrow{h_{t-1}}). \tag{51}$$

The forward LSTM computes the hidden state $\overrightarrow{h_t}$ at time step t based on the input x_t and the previous hidden state $\overrightarrow{h_{t-1}}$.

Backward LSTM

$$\overleftarrow{h_t} = \text{LSTM}_{\text{backward}}(x_t, \overleftarrow{h_{t+1}}). \tag{52}$$

The backward LSTM computes the hidden state $\overleftarrow{h_t}$ at time step t based on the input x_t and the next hidden state $\overleftarrow{h_{t+1}}$.

Concatenated output

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]. \tag{53}$$

The final output h_t at each time step t is the concatenation of the forward hidden state $\overrightarrow{h_t}$ and the backward hidden state $\overleftarrow{h_t}$.

The BiLSTM model captures both the preceding and subsequent contexts, providing a more comprehensive understanding of the sequential data. This bidirectional capability improves performance in applications such as named entity recognition, machine translation, and other sequential prediction tasks.

3.10. Gated recurrent unit

GRU by Cho et al. (2014) are a type of RNN architecture introduced to simplify the standard LSTM model while retaining its ability to capture long-term dependence in sequential data. GRU combine the forget and input gates into a single update gate and merge the cell state and hidden state, leading to a more streamlined structure with fewer parameters.

The key components of a GRU are the update gate and the reset gate, which manage the flow of information through the unit. The GRU mechanism can be described by the following equations:

Update gate

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z). \tag{54}$$

The update gate z_t determines how much of the previous hidden state h_{t-1} needs to be carried forward to the current hidden state. Here, W_z is the weight matrix, x_t is the input at the current time step, h_{t-1} is the hidden state from the previous time step, b_z is the bias term, and σ is the activation function of the sigmoid.

Reset gate

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r). \tag{55}$$

The reset gate r_t determines how much of the previous hidden state should be forgotten. W_r is the weight matrix and b_r is the bias term.

Candidate activation

$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h). \tag{56}$$

The candidate activation $\tilde{h}_t$ is computed using the reset gate r_t. Here, $*$ denotes the Hadamard product (element-wise multiplication), W_h is the weight matrix and b_h is the bias term. This candidate activation is influenced by the reset gate, which modulates the contribution of the previous hidden state.

Hidden state update

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t. \tag{57}$$

The final hidden state h_t is a combination of the previous hidden state h_{t-1} and the candidate activation $\tilde{h}_t$, weighted by the update gate z_t. This equation reflects the balance between retaining the old information and incorporating new information.

The GRU's streamlined architecture, with fewer gates and parameters than an LSTM, often allows for faster training and similar performance on various tasks involving sequential data, such as language modeling and time-series forecasting. The data flow can be visualized as Fig. 19.

3.11. Attention block

The attention mechanism, introduced by Vaswani et al. (2017), represents a significant advancement in the field of deep learning, particularly in natural language processing (NLP). This mechanism facilitates the modeling of dependence without regard to their distance in the input or output sequences. The Transformer model, which is built entirely on attention mechanisms, has become the backbone of various state-of-the-art models in NLP, Seq2Seq, and beyond. The study provides a detailed exploration of the attention block, including its theoretical foundations, mathematical formulation, and practical implications.

Self-attention mechanism

The core component of the Transformer model is the self-attention mechanism, which allows the model to weigh the importance of different words in a sequence relative to each other. This is achieved through the calculation of attention scores, which determine how much focus to place on other parts of the input sequence when encoding a particular part.

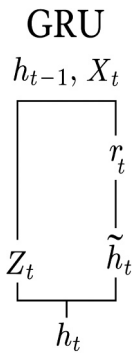


Fig. 19. Visualization of GRU.
Note: Fig. 19 illustrates the structure of a GRU cell. The GRU receives the previous hidden state h_{t-1} and the current input X_t. The update gate Z_t and reset gate r_t control the flow of information, while the candidate hidden state $\tilde{h}_t$ and the resulting hidden state h_t enable efficient modeling of sequential data with fewer parameters compared to LSTM.

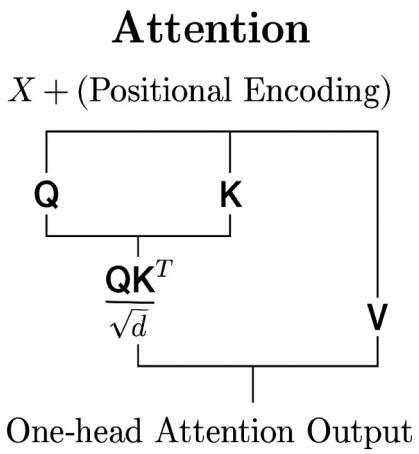


Fig. 20. Visualization of attention.
Note: Fig. 20 illustrates the structure of a single-head attention mechanism. The input X (with positional encoding) is projected to query (Q), key (K), and value (V) representations. The attention weights are computed as the scaled dot product of Q and K, and are then applied to V to produce the output. This mechanism allows the model to focus on relevant parts of the input sequence.

Scaled dot-product attention

The self-attention mechanism in the Transformer model is based on scaled dot-product attention. Given a query matrix Q, a key matrix K, and a value matrix V, the attention mechanism computes a weighted sum of the values, where the weights are determined by the compatibility of the queries and keys. The scaled dot-product attention is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{58}$$

Here, d_k represents the dimension of the keys. Dividing by $\sqrt{d_k}$ is a scaling factor that helps mitigate the effect of having large values of the dot product, which can cause the gradients to become small during training. The data flow can be visualized as Fig. 20.

Multi-head attention

To enhance the model's ability to focus on different positions, Vaswani et al. (2017) introduce the concept of multi-head attention. Instead of performing a single attention function, the multi-head attention mechanism runs multiple attention functions in parallel, each with different learnt projections. The outputs of these attention heads are concatenated and projected to form the final output. This can be expressed as follows.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)W^O, \tag{59}$$

where each head is defined as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{60}$$

Here, W_i^Q, W_i^K, W_i^V, and W^O are learnt projection matrices.

Position-wise feed-forward networks

Following the multi-head attention mechanism, the Transformer model employs position-wise feedforward networks. These consist of two linear transformations with a ReLU activation in between. The same feed-forward network is applied independently to each position. The position-wise feed-forward network is defined as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2, \tag{61}$$

where W_1 and W_2 are weight matrices and b_1 and b_2 are bias terms.

Positional encoding

Since the Transformer model does not inherently include any recurrence or convolution, it lacks a mechanism to incorporate the order of the sequence. To address this, the authors introduced positional encodings, which are added to the input embeddings at the bottom of the encoder and decoder stacks. The positional encoding vectors are designed to give the model information about the relative or absolute position of the tokens in the sequence. The positional encodings are computed as follows:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \tag{62}$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right), \tag{63}$$

where pos is the position and i is the dimension.

Layer normalization and residual connections

To facilitate the training of deep networks, the Transformer model employs layer normalization and residual connections around each sub-layer, including multi-head attention and position-wise feedforward networks. The output of each sublayer is:

$$\text{LayerNorm}(x + \text{Sublayer}(x)), \tag{64}$$

where $\text{Sublayer}(x)$ is either the multihead attention mechanism or the position-wise feedforward network.

Transformer encoder and decoder

The Transformer model consists of an encoder and a decoder, both of which are stacks of identical layers. Each encoder layer has two sublayers: a multihead self-attention mechanism and a position-wise feedforward network. The decoder layer has an additional sublayer that performs multi-head attention over the encoder's output. The encoder and decoder are defined as follows:

$$\text{EncoderLayer}(x) = \text{LayerNorm}(x + \text{FFN}(\text{MultiHead}(x, x, x))), \tag{65}$$

$$\text{DecoderLayer}(y, e) = \text{LayerNorm}(y + \text{FFN}(\text{MultiHead}(y, y, y) + \text{MultiHead}(y, e, e))), \tag{66}$$

where y is the decoder input and e is the encoder output.

The attention mechanism, particularly in the context of the Transformer model, has revolutionized the way sequential data is processed in deep learning. By allowing the model to flexibly weigh the importance of different parts of the sequence, the attention mechanism allows for better handling of long-range dependence and parallelization during training.

3.12. Backpropagation and gradient descent

Backpropagation is a supervised learning algorithm used to train artificial neural networks. It is a method to minimize the error function by adjusting the weights of the network through gradient descent. This algorithm was popularized by Rumelhart et al. (1986) and has since become a cornerstone of deep learning techniques. The core idea behind backpropagation is to propagate the error from the output layer back through the network layers, allowing each layer to adjust its weights to reduce the overall error. The mathematical foundation of backpropagation is grounded in calculus, specifically in the chain rule, which is used to compute the gradients of the error function with respect to the weights.

The process begins with a forward pass through the network, where inputs are transformed into outputs through a series of weight multiplications and activation functions. The output is compared to the target and an error is calculated. This error is then propagated backward through the network in the backward pass, where the weights are adjusted to minimize the error. Mathematically, let **x** be

the input vector, $\mathbf{y}$ the target vector, $\hat{\mathbf{y}}$ the output vector, $\mathbf{w}$ the weight vector and L the loss function. The objective is to minimize $L(\mathbf{y}, \hat{\mathbf{y}})$. During backpropagation, the loss function gradient is calculated with respect to each weight, $\frac{\partial L}{\partial \mathbf{w}}$, and used to update the weights.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial L}{\partial \mathbf{w}}, \tag{67}$$

where η is the learning rate. The backpropagation algorithm can be summarized in the following steps:

a. Initialization: Initialize the weights randomly.
b. Forward pass: Compute the output of the network for a given input by passing it through each layer.
c. Compute error: Calculate the error at the output layer using a loss function such as mean squared error or cross-entropy.
d. Backward pass: Propagate the error through the network:

1. Calculate the gradient of the loss function with respect to the output of the network.
2. Use the chain rule to compute the gradient of the loss function with respect to the weights of each layer.
3. Update the weights using the computed gradients.

e. Repeat: Repeat the forward and backward passes for a number of epochs or until the error converges to a minimum value.

Backpropagation is a powerful and essential algorithm for training neural networks. Its ability to iteratively adjust the weights to minimize the loss function has enabled the successful deployment of neural networks in numerous real-world applications. The combination of mathematical rigour and practical efficiency makes backpropagation a foundational technique in the field of ML.

3.13. Loss function

A loss function, also known as a cost function or an objective function, is a mathematical formulation utilized to quantify the discrepancy between the predicted outputs of a model and the actual target values. It serves as a fundamental component in the optimization of ML algorithms, providing a measure of the model's predictive performance. The primary objective in training an ML model is to minimize this loss function, thereby improving the accuracy and generalization capabilities of the model.

Loss functions can be broadly categorized into different types, each suited for specific tasks and types of data. For regression tasks, common loss functions include the mean squared error (MSE), the mean absolute error (MAE), and Huber loss. The MSE, for instance, is defined as the average of the squared differences between the predicted values and the actual values:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{68}$$

where n represents the number of data points, y_i denotes the actual value and $\hat{y}_i$ denotes the predicted value. The squaring of differences in MSE emphasizes larger errors, making it sensitive to outliers.

3.14. Asynchronous successive halving algorithm

AutoML, or Automated ML, represents a transformative approach within the field of AI and data science, aiming to automate the end-to-end process of applying ML to real-world problems. Traditional ML workflows require extensive human intervention in various stages, including data preprocessing, feature engineering, model selection, hyperparameter tuning, and model evaluation. AutoML addresses these complexities by leveraging sophisticated algorithms and computational techniques to streamline and enhance these processes, making ML more accessible and efficient.

The core advantage of AutoML lies in its ability to democratize ML, allowing individuals with limited expertise in the domain to develop competitive models. This is achieved through the integration of various methodologies such as neural architecture search (NAS), automated feature engineering, and meta-learning. The search for neural architecture, for example, automates the design of neural network structures, optimizing them for specific tasks without human intervention. Automated feature engineering, another critical component, systematically transforms raw data into meaningful features that improve model performance. Meta-learning, or "learning to learn'' employs previous learning experiences to expedite the training of new models. Furthermore, AutoML systems often incorporate advanced optimization techniques like Bayesian optimization or the asynchronous successive halving algorithm (ASHA) proposed by Li et al. (2020b) to fine-tune model hyperparameters efficiently. In academic research, AutoML facilitates the handling of large-scale, complex datasets, enabling researchers to focus more on theoretical advancements and less on the intricacies of model building. The advent of AutoML marks a significant milestone in the evolution of ML, promising to accelerate innovation and enhance the practical deployment of AI technologies across diverse domains.

The ASHA is a hyperparameter optimization method designed for large-scale parallelism by leveraging aggressive early stopping. This method builds upon the Successive Halving Algorithm (SHA), which is enhanced to handle distributed computing environments effectively. ASHA addresses the issue of needing to evaluate a vastly larger number of hyperparameter configurations than there are available parallel workers, achieving this within a small multiple of the wall-clock time required to train a single model.

The ASHA methodology revolves around the incremental growth of brackets and the use of intermediate losses to identify the best-performing configurations more quickly, compared to the fixed budget intervals of SHA. This method significantly reduces latency by using these intermediate metrics rather than waiting for the final SHA output to be completed. Furthermore, ASHA does not require prespecification of the number of configurations to evaluate, offering flexibility and efficiency in hyperparameter tuning.

Table 2
Hyperparameters optimized by ASHA for Regression Tree Models.

Hyperparameter	Description	Range/Options
MaxNumSplits	Maximum number of splits (nodes) in the tree.	Integer values, e.g. from 10 to 1000
MinLeafSize	Minimum number of observations per leaf (terminal node).	Integer values, e.g. from 1 to 100
MinParentSize	Minimum number of observations required to create a new split in the tree.	Integer values, often starting from 2 and increasing as needed
SplitCriterion	Criterion were used to choose the best split at each node.	ǧdi', 'deviance', etc.
Surrogate	Whether to use surrogate splits to handle missing data.	'on' or 'off'

Table 3
Hyperparameters optimized by ASHA for Ensemble Models.

Hyperparameter	Description	Range/Options
Method	The type of ensemble method used.	'Bag', 'LSBoost', 'RUSBoost', etc.
NumLearningCycles	The number of learning cycles (iterations).	Integer values, e.g. from 10 to 1000
LearnRate	The learning rate for boosting algorithms.	Positive real numbers, for example, from 0.01 to 1
MaxNumSplits	The maximum number of splits (nodes) in each decision tree.	Integer values, e.g. from 10 to 1000
MinLeafSize	The minimum number of observations per leaf (terminal node).	Integer values, for example, from 1 to 100
NumVariablesToSample	The number of predictors to sample at each node when growing trees.	Integer values, e.g., from 1 to the total number of predictors

(Li et al., 2020b) demonstrate that ASHA outperforms several state-of-the-art hyperparameter optimization methods such as Population-Based Training (PBT), Bayesian Optimization Hyperband (BOHB), and Google's Vizier. Through extensive empirical evaluations on tasks involving CIFAR-10 and other benchmarks, ASHA consistently shows better performance, particularly excelling in environments with high variance in training times between configurations. It also scales linearly with the number of workers, thus proving its robustness and effectiveness in massively parallel settings.

According to the study by Du Bois et al. (2023), this approach has made significant contributions to the optimization of time series forecasting models, which is used during training phases in a 10-fold cross-validation framework, providing efficient hyperparameter optimization. Their work demonstrates improved model selection and training efficiency, leading to better generalization and performance of the models by implementing ASHA. It allows for concurrent processing and dynamic resource allocation, which ensures that the most promising models receive the necessary computational resources early on, thus enhancing the overall effectiveness of the time series forecasting tasks evaluated in the study.

In this paper, ASHA is employed to optimize the hyperparameters of Regression Tree, Ensemble, Kernel, Linear, SVR, NN, GP models for model specifications. The details are given in Tables 2 to 8.

4. Data, test designs and empirical results

4.1. Data generating process

To implement the first test, we use the generator to obtain a long series consisting of 12,512 observations drawn from the standard normal distribution where the mean is 0 and the standard deviation is 1. Then we employ a fractional differencing procedure to obtain the long-memory series with $d = 0.3$ and sample the data with a 512-length rolling window with a step size of 1. Thus, the final training dataset for ML is a matrix containing 11,488 rows and 512 columns with long-memory parameter d, which is equal to 0.3. The desired output of ML is to predict the next value of each series of columns. In the training phase, the inputs, outputs of ML and real values can be visualized as Table 9. In the testing phase, at the start, the last 512 observations of the time series, $X_{11489}, X_{11490}, \ldots, X_{12000}$, are given to the ML models as inputs to predict the value of X_{12001}. In the next step, the inputs are shifted by one unit, which means that well-trained models use the real values $X_{11490}, X_{11491}, \ldots, X_{12001}$ to generate X_{12002}. This process continues until we obtain a new time series with 512 observations, which is used to estimate its long-memory parameter d compared to the real series with $d = 0.3$. For systemic memory examination, the data generated by the method of Shimotsu (2012) contain 12,000 observations, but each observation includes two series, X and Z have 512 data points, respectively, where X and Z have the fractional cointegration relation. The memory parameter of X is equal to 0.3, while the memory parameter of $Z - CX$

Table 4
Hyperparameters optimized by ASHA for Kernel Models.

Hyperparameter	Description	Range/Options
KernelFunction	Type of kernel function used to transform the input data.	'linear', 'gaussian', 'polynomial', 'rbf', etc.
BoxConstraint	Regularization parameter C, controlling the trade-off between achieving a low training error and a low testing error.	Positive real numbers, for example, from 0.1 to 1000
KernelScale	Scaling factor for the kernel function, particularly important for RBF and Gaussian kernels.	positive real numbers, for example, from 0.1 to 100
Epsilon	Epsilon-insensitive loss function parameter, defining a tolerance margin where no penalty is given for errors within this margin.	Positive real numbers, for example, from 0.01 to 1
Standardize	Flag indicating whether predictor data should be standardized (mean-centred and scaled to unit variance).	true or false

Table 5
Hyperparameters optimized by ASHA for Linear Models.

Hyperparameter	Description	Range/Options
Intercept	Whether to include an intercept term in the model.	true or false
RobustOpts	Options for robust fitting to handle outliers.	'on' or 'off'
Lambda	regularization parameter for ridge regression (L2 regularization).	positive real numbers, for example, from 0.1 to 10
Alpha	Elastic Net Mixing Parameter.	real number between 0 (ridge) and 1 (lasso)
Standardize	Whether to standardize the predictor variables.	true or false

Table 6
Hyperparameters optimized by ASHA for SVR Models.

Hyperparameter	Description	Range/Options
KernelFunction	Type of kernel function used to transform the input data.	'linear', 'gaussian', 'polynomial', 'rbf', etc.
BoxConstraint	Regularization parameter C, controlling the trade-off between achieving a low training error and a low testing error.	Positive real numbers, for example, from 0.1 to 1000
KernelScale	Scaling factor for the kernel function, particularly important for RBF and Gaussian kernels.	positive real numbers, for example, from 0.1 to 100
Epsilon	Epsilon-insensitive loss function parameter, defining a tolerance margin where no penalty is given for errors within this margin.	Positive real numbers, for example, from 0.01 to 1
Standardize	Flag indicating whether predictor data should be standardized (mean-centred and scaled to unit variance).	true or false

is equal to 0.1, where C is a constant set to 1. It is noted that, for MLP models, we convert the observation structure to the vector so as to adjust the MLP models.

To implement the second test, we use the same generator and fractional differencing procedure to obtain the training dataset, which contains 13024 observations with various values of d ranging from -0.4 to 1. Each observation includes 512 data points corresponding to a specific d value. In the training phase, we employ ML models to learn the mapping between the first 12,000 observations and d, while well-trained models predict the last 1024 d with the rest of the observations. We set the error threshold between the predicted values and the estimated values by FELW at 0.1, calculating the percentage of d that falls in the accepted interval. Similarly, for systemic memory, we apply a fractional differencing procedure to get two fractionally cointegrated series

Table 7
Hyperparameters optimized by ASHA for Neural Network Models.

Hyperparameter	Description	Range/Options
LayerSizes	The sizes of the fully connected layers.	Vector of positive integers, e.g., [10, 50, 100]
Activations	Activation functions for each layer.	String array of cells, for example 'relu', 'tanh', 'sigmoid'
LearnRate	The learning rate for training the neural network.	Positive real numbers, for example, from 0.001 to 1
Regularization	The regularization strength to prevent overfitting.	Positive real numbers, for example, from 0.0001 to 1
Solver	The optimization algorithm to train the network.	'sgdm', 'adam', rmsprop', etc.
Standardize	Whether to standardize the predictor variables.	true or false

Table 8
Hyperparameters optimized by ASHA for Gaussian Process Regression Models.

Hyperparameter	Description	Range/Options
KernelFunction	Type of kernel function used to compute the covariance.	'squaredexponential', 'matern32', 'matern52', 'rationalquadratic', etc.
KernelScale	Scaling factor for the kernel function.	Positive real numbers, for example, from 0.1 to 10
BasisFunction	Type of basis function used for the regression.	'constant', 'linear', 'pureQuadratic', etc.
Sigma	Noise standard deviation.	Positive real numbers, for example, from 0.01 to 1
Standardize	Whether to standardize the predictor variables.	true or false

Table 9
Inputs and Outputs of ML training phase compared to Desired Output or Real Value with N=12000.

Inputs of ML	Outputs of ML	Desired output or real value
$X_1, X_2, \ldots, X_{512}$	X'_{513}	X_{513}
$X_2, X_3, \ldots, X_{513}$	X'_{514}	X_{514}
$X_3, X_4, \ldots, X_{514}$	X'_{515}	X_{515}
$\vdots$	$\vdots$	$\vdots$
$X_{N-512}, X_{N-511}, \ldots, X_{N-1}$	X'_N	X_N

Table 10
Inputs and Outputs of Seq2Seq training phase compared to Desired Output or Real Value with N=12000.

Inputs of ML	Outputs of ML	Desired Output or Real Value
Y_1	d'_1	d_1
Y_2	d'_2	d_2
Y_3	d'_3	d_3
$\vdots$	$\vdots$	$\vdots$
Y_N	d'_N	d_N

with various memory parameters ranging from 0.1 to 0.9. Thus, each observation is a matrix with 512 rows and 2 columns, and the output is a vector including the estimated constant and two memory parameters presented in Table 10.

We collect the minutely closing prices of BTC denominated in USD, sourced via the Binance exchange Application Programming Interface (API). Bitcoin is selected as the focal asset due to its pre-eminent status as the first and largest cryptocurrency by market capitalization. The data span from January 1, 2021, through October 18, 2024, yielding a total of 1,309,508 observations.

To mitigate computational burden and account for potential microstructure noise, we downsample the original series by selecting one observation every 87 min, resulting in 15,052 equally spaced price points. The transformed price series is then converted into

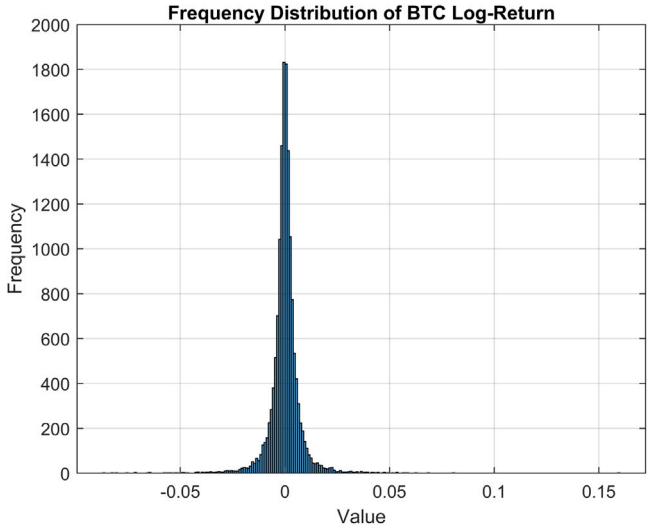


Fig. 21. Bitcoin Log-Return.
Note: Fig. 21 presents the frequency distribution (histogram) of Bitcoin log-returns. The distribution is sharply peaked around zero and exhibits heavy tails, indicating the presence of extreme values more frequently than would be expected under a normal distribution. This feature is commonly observed in high-frequency financial return data.

Table 11
Descriptive statistics of log returns.

Statistic	Value	Statistic	Value
Mean	5.59×10^{-5}	Median	3.99×10^{-5}
Mode	0	Standard Deviation	0.00732
Variance	5.35×10^{-5}	Minimum	−0.08687
Maximum	0.15948	1st Quartile (Q1)	−0.00236
3rd Quartile (Q3)	0.00250	Interquartile Range (IQR)	0.00486
Skewness	0.316	Kurtosis	34.87

logarithmic returns as follows:

$$r_t = \log(P_t) - \log(P_{t-1}), \tag{69}$$

where P_t denotes the BTC/USD price at time t. A rolling-window method is employed for predictive modeling, wherein each input comprises a 512-step sequence of past log-returns, and the target variable corresponds to the one-step-ahead return. The rolling window is updated with an increment of one observation, thus allowing for dynamic temporal modeling across the entire series.

The empirical distribution of Bitcoin log-returns is illustrated in Fig. 21.

Table 11 reports the descriptive statistics of the log return series, highlighting its central tendency, dispersion, and higher moments. The mean of the log returns is close to zero, which is typical for high-frequency financial return series. The standard deviation of 0.00732 reflects considerable volatility. Notably, the minimum and maximum returns, −8.69% and 15.95% respectively, suggest the presence of extreme observations.

The skewness is moderately positive (0.316), indicating a slightly longer right tail; however, this value is relatively close to zero, implying near symmetry. In contrast, the kurtosis is extremely high at 34.87, substantially exceeding the normal distribution's kurtosis of 3, evidencing the presence of heavy tails and extreme events, a well-known feature of financial return distributions.

These statistics collectively underscore the non-Gaussian characteristics of the series, namely leptokurtosis and slight asymmetry. Such properties are congruent with empirical findings in financial econometrics and justify the adoption of fat-tailed or stable distributions, such as those discussed in the context of fractionally differenced sequences and heavy-tailed models (e.g., the Lévy stable, Student-t, or generalized hyperbolic families).

The pronounced excess kurtosis ($\kappa = 34.87$) and moderate positive skewness ($\gamma = 0.316$) deviate significantly from the assumptions of normality and call into question the applicability of Gaussian models for the log return process. These empirical findings are consistent with the stylized facts in financial econometrics literature, where return distributions are often found to be leptokurtic and mildly skewed. Given these features, the normal distribution is inadequate to capture the empirical distribution of returns.

The descriptive statistics reveal a log return process with negligible mean, moderate volatility, slight skewness, and substantial excess kurtosis. These features are incompatible with normality and instead support the application of fat-tailed distributions such as the Student-t, Generalized Hyperbolic, or Lévy Stable laws. The memory parameter d estimated by the FELW method is

Table 12
Specifications of seven classical models for one-step prediction.

Specification	Ensemble	GP	Kernel	Linear	Net	SVM	Tree
Objective	0.71919	0.77095	0.77453	0.71175	0.73311	0.73817	0.73522
Objective Evaluation Time	89.224	13.568	9.5676	0.44337	6.4979	21.002	1.5802
Training Set Size	2400	600	2400	2400	2400	2400	2400
Method/Function	LSBoost	rationalquadratic	leastsquares	svm	sigmoid	linear	–
Num Learning Cycles	58	–	–	–	–	–	–
Learn Rate	0.053607	–	–	–	–	–	–
Min Leaf Size	2	–	–	–	–	–	69
Max Num Splits	4	–	–	–	–	–	914
Num Variables to Sample	NaN	–	–	–	–	–	351
Kernel Scale	–	5.1766	54.444	–	–	NaN	–
Lambda	–	–	0.0012594	0.040108	0.11401	–	–
Epsilon	–	–	NaN	–	–	0.23061	–
Basis Function	–	none	–	–	–	–	–
Standardize	–	true	–	–	false	true	–
Sigma	–	0.1438	–	–	–	–	–
Regularization	–	–	–	lasso	–	–	–
Num Expansion Dimensions	–	–	2696	–	–	–	–
Num Layers	–	–	–	–	1	–	–
Layer Weights Initializer	–	–	–	–	glorot	–	–
Layer Biases Initializer	–	–	–	–	zeros	–	–
Layer 1 Size	–	–	–	–	8	–	–
Box Constraint	–	–	–	–	–	0.0014399	–
Polynomial Order	–	–	–	–	–	NaN	–

approximately 0.03, suggesting that the BTC return series exhibits behavior close to market efficiency, as values of d near zero are indicative of short memory. To assess the practical feasibility and robustness of the proposed models under varying degrees of memory, we apply fractional differencing to the return series to synthetically induce long-memory characteristics. Specifically, we transform the series such that the differencing parameter is set to $d = 0.3$, thereby introducing persistent autocorrelation in the return dynamics while preserving the heavy-tailed nature inherent in financial time series. This transformation allows for evaluating model performance in a controlled long-memory environment.

4.2. Empirical results

We first use the ASHA algorithm to optimize the hyperparameters and train seven classical models including RF, GP, Kernel, Linear, MLP, SVR, and RT. Then we use these seven well-trained models to make 512-step predictions with both open-loop forecasting and closed-loop forecasting strategies. We estimate the d value of the predicted series using the FELW approach. The predicted series should have the same statistical properties as the original series, where the value d should be equal to 0.3. As for Seq2Seq models, we have an augmented test to examine their ability to capture systemic memory representing cointegration relations.

Type I test: Learning results of seven models for univariate memory

Univariate memory describes whether a time series tends to trend, mean-revert, or follow a random walk. It can effectively reflect the speed of information digestion, indicating the exploitable potential of the corresponding strategies. To examine whether the RF, GP, Kernel, Linear, MLP, SVR and RT models can capture long-memory property from the data, we use ASHA to specify the model hyperparameters and estimate the values of memory parameters for both training dataset and predicted series by the FELW estimator. The model specifications are presented in Table 12.

The results are shown in Table 13 compared to the real value $d = 0.3$. We note that the d value of kernel methods is equal to 0.5019, meaning that this approach cannot learn long memory well even in the training dataset. Moreover, almost all models have poor processing capability for long-range dependence because the predicted series have large differences from $d = 0.3$, where the accepted interval is set from 0.2 to 0.4. The results of closed-loop prediction by kernel, linear, MLP and RT models have values close to 0.3, indicating that these models might have intrinsic potential to generate long-memory series from learning behaviors. However, these abilities are fragile because the models cannot reveal the real degree of long memory when it comes to exogenous noise. Interestingly, the simple RT model outperforms other models, suggesting complicated models such as RF are not always the better choice in processing long-memory time series.

Type I test: Learning results of Seq2Seq models for univariate memory

By comparison, the results of the Seq2Seq models for univariate memory are shown in Table 14. We note that the original series has a significant long memory with $d = 0.3$. If Seq2Seq models can learn long memory, the predicted series generated by these

Table 13
Univariate memory parameter of series predicted by various models under training and testing scenarios.

Dataset d=0.3	Train open	Test open	Test closed
RF	0.3036	0.4870	0.7788
GP	0.2799	0.5095	0.7692
Kernel	0.5019	0.5012	0.3628
Linear	0.3037	0.4943	0.2971
MLP	0.3991	0.5407	0.3787
SVR	0.3458	0.4660	0.1488
RT	0.3353	0.3921	0.2215

Table 14
Univariate memory parameter of series predicted by Seq2Seq models.

d=0.3	Open	Closed
LSTM	0.3271	0.3932
BiLSTM	0.3425	0.4134
GRU	0.3217	0.4766
Attention	0.4843	0.6247
Attention-LSTM	0.3738	0.3612
Embedding-Attention-LSTM	0.3332	0.3070
MLP-Embedding-Attention-LSTM	0.2904	0.3540

Table 15
Performance comparison of machine learning and Seq2Seq models.

Model	Test Open/Open	MSE	Accuracy
Embedding-Attention-LSTM	0.3332	1.4561	0.58708
GRU	0.3217	1.4175	0.57339
LSTM	0.3271	1.4265	0.56751
RT (Tree)	0.3921	1.2478	0.56947
Attention-LSTM	0.3738	1.4665	0.56947
BiLSTM	0.3425	1.4094	0.54403
MLP-Embedding-Attention-LSTM	0.2904	1.4496	0.54403
SVR (SVM)	0.4660	1.0747	0.51663
Kernel	0.5012	1.1894	0.51663
GP	0.5095	1.0602	0.51076
MLP (Net)	0.5407	1.0757	0.50685
Attention	0.4843	1.1568	0.48141
RF (Ensemble)	0.4870	1.0572	0.45597
Linear	0.4943	1.0362	0.44031

Note: Test Open/Open measures the proximity to the ideal long memory parameter ($d \approx 0.3$), MSE denotes mean squared error, and Accuracy represents directional prediction accuracy.

well-trained models should also have a similar long memory property. With open-loop forecasting strategies, MLP-Embedding-Attention-LSTM outperforms other models, while the Embedding-Attention-LSTM model has the best performance in closed loop prediction. Seq2Seq models except the Attention model generally have higher predictive accuracy than the seven classical models do under both open and closed looping forecasting strategies. The results also show that hybrid models generally outperform single models, echoing the studies by Li et al. (2020a), Júnior et al. (2019), Nosratabadi et al. (2020), Ciaburro and Iannace (2021), Yan and Ouyang (2018). Consequently, Seq2Seq models, particularly the embedded-attention-LSTM models, are proved to have great potential in processing series prediction with univariate long-range dependence.

The results in Table 15 reveal an important limitation of relying solely on MSE for model selection. While traditional machine learning models such as Linear, RF (Ensemble), GP, and SVR achieve relatively low MSE values, they do not exhibit high accuracy in predicting the direction of returns. In contrast, models with Test Open (or Open) values closer to 0.3, such as Embedding-Attention-LSTM, GRU, and LSTM, demonstrate significantly higher accuracy in forecasting market trends, despite their higher MSE.

This evidence highlights that models capable of capturing long-memory characteristics, notably sequence-to-sequence architectures and their variants, provide substantial advantages for directional trend prediction in cryptocurrency markets. In other words, the ability to preserve long-term dependence, rather than simply minimizing pointwise prediction error, proves to be more important for the economic task of forecasting price movement directions.

Overall, the proposed methodology, which prioritizes the long-memory capturing ability, demonstrates clear superiority over conventional MSE-based selection. This approach provides meaningful guidance for both model evaluation and the practical deployment of forecasting models in financial trend prediction.

Table 16
Systemic memory parameters of two series predicted by Seq2Seq models.

Forecasting strategies	Open			Closed		
C=1, $D_1 = 0.1$, $D_2 = 0.3$	C	D_1	D_2	C	D_1	D_2
LSTM	0.9994	0.1226	0.3816	0.8147	0.0738	0.4371
BiLSTM	−382.8008	0.1706	0.1708	0.8404	0.1462	0.4222
GRU	13.9153	0.2857	0.2832	0.8046	0.0882	0.4346
Attention	12.5263	0.3544	0.3486	0.8905	0.1326	0.3372
Attention-LSTM	0.9481	0.1022	0.3538	0.8547	0.1117	0.4416
Embedding-Attention-LSTM	0.9858	0.098	0.3484	0.9812	0.0455	0.4559
MLP-Embedding-Attention-LSTM	0.9108	−0.0044	0.2893	1.3308	0.0882	0.3731

Note: With respect to the memory parameters estimated by FCELW (Shimotsu, 2012), the values are presented as $C = 0.8565$, $D_1 = 0.1658$, $D_2 = 0.3812$.

Table 17
Specifications of seven classical models for one-step prediction.

Specification	Ensemble	GP	Kernel	Linear	Net	SVM	Tree
Objective	0.041778	0.040292	0.049196	0.15072	0.031946	0.048911	0.048169
Objective Evaluation Time	42.292	4.2309	6.1282	0.22574	60.155	17.311	6.1949
Training Set Size	2400	600	150	600	2400	2400	2400
Method/Function	Bag	exponential	leastsquares	svm	tanh	gaussian	–
Num Learning Cycles	13	–	–	–	–	–	–
Learn Rate	NaN	–	–	–	–	–	–
Min Leaf Size	7	–	–	–	–	–	5
Max Num Splits	1976	–	–	–	–	–	10 472
Num Variables to Sample	225	–	–	–	–	–	357
Kernel Scale	–	52.215	30.271	–	–	0.47536	–
Lambda	–	–	1.7334e−06	0.55271	1.0037e−05	–	–
Epsilon	–	–	NaN	–	–	0.09107	–
Basis Function	–	none	–	–	–	–	–
Standardize	–	true	–	–	true	true	–
Sigma	–	0.053444	–	–	–	–	–
Regularization	–	–	–	ridge	–	–	–
Num Expansion Dimensions	–	–	1614	–	–	–	–
Num Layers	–	–	–	–	2	–	–
Layer Weights Initializer	–	–	–	–	glorot	–	–
Layer Biases Initializer	–	–	–	–	ones	–	–
Layer 1 Size	–	–	–	–	2	–	–
Layer 2 Size	–	–	–	–	8	–	–
Layer 3 Size	–	–	–	–	–	–	–
Box Constraint	–	–	–	–	–	3.5802	–
Polynomial Order	–	–	–	–	–	NaN	–

Type I test: Learning results of Seq2Seq models for systemic memory

Systemic memory, or fractional cointegration, represents an equilibrium relation in which two fractionally integrated series have a linear combination that is (fractionally) stationary. In financial markets, prices with these relations suggest that the difference of prices should finally converge to a reasonable range after encountering external shocks, which is considered as a foundation of pair-trading strategies.

According to Table 16, BiLSTM, GRU and Attention models have weak capability of capturing fractional cointegration relations under open-loop forecasting though the attention mechanism performs well in closed-loop forecasting. LSTM models have the most stable performance among the four single Seq2Seq models. Compared to the LSTM model, the hybrid models involving embedding, attention, and MLP components do not have a significant enhancement of predictive accuracy. Relatively, the Attention-LSTM model has a slight improvement in forecasting sequences with the heterogeneous equilibrium relation in both the open-loop and closed-loop tests.

Type II test: Learning results of seven models for univariate memory

ML is famous for its power in uncovering latent data patterns, predictive ability, and convenience of end-to-end learning. In the spirit of the equivalence principle, if series memory can be identified as the FELW estimator does, these advanced methods are considered to learn long-range dependence well. In parallel, Table 17 shows the model specifications of Seven Classical Models Optimized by the ASHA algorithm.

The seven models learn from a synthetic dataset containing 12,000 time series with various long-memory parameters ranging from −0.4 to 1. We use these well-trained models to recognize the memory of new 1,024 time series and compute ratios of cases with an error of less than 0.1. We consider these ratios as the acceptance rate to evaluate the ability of ML to act as FELW methods,

Table 18
Acceptance rates for various algorithms.

Algorithm	Acceptance rate
RF	0.4141
GP	0.5146
Kernel	0.2715
Linear	0.1504
MLP	0.3203
SVR	0.3799
RT	0.3604

Note: The acceptance rate indicates the proportion of cases where the condition $|d - \hat{d}| < 0.1$ is satisfied out of 1024 samples.

Table 19
d value predictions and acceptance rates for univariate memory under DNN and MLP models with $d = 0.3$.

Models	DNN (Open)	DNN (Closed)	MLP (Open)	MLP (Closed)
Univariate Memory (d=0.3)	0.36287	0.24609	0.54068	0.37871
Acceptance rate for Univariate Memory	0.6045	–	0.32031	–

Table 20
Results of systemic memory estimations under DNN and Seq2Seq models.

Forecasting strategies	Open			Closed		
C=1, $D_1 = 0.1$, $D_2 = 0.3$	C	D_1	D_2	C	D_1	D_2
DNN	1.1989	−0.0119	0.3905	0.5933	−0.0222	0.0817
LSTM	0.9994	0.1226	0.3816	0.8147	0.0738	0.4371
Attention-LSTM	0.9481	0.1022	0.3538	0.8547	0.1117	0.4416
Embedding-Attention-LSTM	0.9858	0.098	0.3484	0.9812	0.0455	0.4559
Acceptance rates by DNN	0.0137	0.2559	0.2705	–	–	–

which is shown in Table 18. GP models achieve the highest acceptance rate of 0.5146, indicating that this model is more accurate compared to other models in estimating the long memory parameter d for the new time series. However, this result is unsatisfied as only 51.46% of the cases have an estimation error of less than 0.1, indicating that the classical ML models cannot be a suitable choice for time series with long memory, which explains why the prediction made by these models in financial markets is invalid.

Comparison of learning results of MLP models for both univariate and systemic memory

To further explore the impact of the number of layers in neural network models, we conducted an augmented test that specifically examines the potential of modeling long-memory time series by varying the depth of the networks. Table 19 presents the results of estimation d using the FELW approach on a time series of 512-length, which was generated by DNN and MLP.

Our findings indicate that increasing the depth of the layers in these neural network models significantly enhances their predictive performance. Specifically, as the number of layers in the DNN and MLP models increases, the models become more capable of capturing the complex, long-term dependence that are characteristic of long-memory processes. This is particularly evident in the comparison between DNN and MLP in both open and closed loop forecasting, where deeper networks consistently produce more accurate d value estimates.

For instance, in the open loop condition, the DNN model, which typically has a deeper architecture than the MLP model, demonstrates a higher acceptance rate for d value predictions that are within the error threshold of 0.1. This suggests that deeper models are better at generalizing across different data regimes, leading to improved performance not only in terms of accuracy, but also in consistency across varying conditions.

In summary, the results underscore the importance of layer depth in neural network architectures when modeling complex time series data. The increased depth allows the network to learn more intricate representations of the data, thereby improving the estimation of long-memory parameters such as d. These findings align with the broader understanding that deeper neural networks, with their ability to capture hierarchical features, are more effective for complex predictive tasks, especially in the context of long-memory time series analysis.

Finally, the comparison of estimating systemic memory parameters between DNN, Seq2Seq models, and hybrid models is presented in Table 20. This table summarizes the results of the estimation of systemic memory under both Open and Closed conditions, with a focus on the systemic memory parameters C, D_1, and D_2, where the true values are $C = 1$, $D_1 = 0.1$, and $D_2 = 0.3$.

In the Open condition, the DNN model estimates C as 1.1989, which slightly overestimates the true value of 1, suggesting that the DNN model may be slightly sensitive to the data structure when estimating the linear coefficient C. The estimated values for D_1 and D_2 are −0.0119 and 0.3905, respectively. Here, the estimate for D_1 deviates significantly from the true value of 0.1, underestimating it, while the estimate for D_2 is slightly above the true value of 0.3.

Table 21
d estimation for predictive data series and real data by FELW.

Bandwidth	Original data	DNN	LSTM	Attention	AttentionLSTM	EmbedAttentionLSTM
0.5	0.22887	0.43451	0.44554	−0.16008	0.19597	0.37026
0.6	0.22611	0.26183	0.39960	0.19004	0.19500	0.43476
0.7	0.30396	0.29329	0.46100	0.16651	0.32337	0.44100

The LSTM model demonstrates a closer estimate for C at 0.9994, which is almost identical to the true value, indicating a strong predictive performance in estimating C. The estimates for D_1 (0.1226) and D_2 (0.3816) are also closer to the true values compared to DNN, although still showing some deviation, particularly in the estimate D_2, which overestimates the true value.

The Attention-LSTM model provides an estimate for C of 0.9481, which underestimates the true value of 1, suggesting a modest bias. However, its estimates for $D_1 = 0.1022$ and $D_2 = 0.3538$ are closer to the true values of 0.1 and 0.3, respectively, with only a mild overestimation in D_2. This suggests it has the strongest fidelity to the true parameter values among all evaluated models under the *Open* condition.

The Embedding-Attention-LSTM does not further improve estimation accuracy for $D_1 = 0.098$, although $C = 0.9858$ is an almost precise match to the true value. The estimates for $D_2 = 0.3484$ also align closely with the actual values. Its performance suggests that incorporating both positional encoding (embedding) and attention mechanisms may not enhance the performance of capturing the systemic long memory.

In terms of acceptance rates by the DNN model, the values are 0.0137 for $C = 1$, 0.2559 for $D_1 = 0.1$, and 0.2705 for $D_2 = 0.3$, indicating a considerably lower robustness in recognizing systemic long memory.

Under the Closed condition, the DNN model estimates C at 0.5933, which substantially underestimates the true value of 1. This divergence suggests a potential model mis-specification or an inability of the DNN architecture to fully capture the conditional dynamics in the presence of closure constraints. The estimates for $D_1 = -0.0222$ and $D_2 = 0.0817$ also deviate notably from the true values of 0.1 and 0.3, with D_1 turning negative and D_2 significantly underestimated. These patterns point to a diminished ability of the DNN to infer long-memory features in this regime.

In contrast, the LSTM model offers much more accurate estimates. The coefficient C is estimated at 0.8147, closer to the true value, though still displaying a downward bias. Its estimates for $D_1 = 0.0738$ and $D_2 = 0.4371$ suggest mild underestimation of D_1 and overestimation of D_2. While not perfect, these figures are considerably more consistent with the underlying parameter structure than those of the DNN.

The Attention-LSTM continues to improve upon this trend. It estimates $C = 0.8547$, again underestimating the true value, but within a tolerable margin. The memory parameters $D_1 = 0.1117$ and $D_2 = 0.4416$ are both slightly overestimated but demonstrate the model's capacity to recognize persistent dependence within the series.

Finally, the Embedding-Attention-LSTM yields the most accurate estimates under the *Closed* condition: $C = 0.9812$, $D_1 = 0.0455$, and $D_2 = 0.4559$. The linear term closely matches the ground truth, confirming the model's robustness. However, D_1 is underestimated and D_2 is overestimated, which may reflect a trade-off in the model's learning behavior between short and long memory features.

In general, this analysis highlights the varying degrees of accuracy in the estimation of systemic memory among different architectures. Although DNN shows stronger performance in some aspects, the Seq2Seq LSTM model generally provides more accurate estimates for the systemic memory parameters C, D_1, and D_2. The Hybrid model, while showing promise, particularly in closed conditions, also exhibits variability in its estimations. These findings suggest that, while deeper or more complex models can capture intricate patterns in the data, their performance may still be sensitive to the underlying conditions, emphasizing the importance of model selection and tuning for accurate systemic memory estimation.

Table 21 reports the estimates of the memory parameter d across predictive models and real BTC data under various bandwidths, based on the FELW method. Despite the potential complexity and deviation of BTC log-returns from classical distributional assumptions, the AttentionLSTM model consistently demonstrates superior performance across all bandwidths, aligning most closely with the d value of the real data. Surprisingly, the DNN model, despite its simplicity, ranks second and outperforms both LSTM and Attention individually. The underperformance of the more sophisticated Embedding Attention-LSTM model suggests that architectural complexity and ensemble learning do not necessarily guarantee robustness. Moreover, the individual LSTM and Attention architectures exhibit less stable performance, highlighting the sensitivity of these models to underlying data distributions, especially in environments characterized by regime-switching behavior. These findings underscore the importance of accounting for distributional properties when developing forecasting models for cryptocurrency markets.

5. Conclusion

Motivated by the consistency of statistical properties across varying distributions, we revisit the foundational question regarding the role of long memory in financial return series. While long memory has long been associated with market inefficiencies, our findings uncover a novel implication: long memory in returns might facilitate heavy-tailed behavior in price distributions, irrespective of the originating distribution, ranging from stable to skewed and generalized forms. This observation implies that long memory may exacerbate extreme risk exposure in financial markets.

Furthermore, we empirically validate the robustness of the FELW estimator using samples drawn from a wide spectrum of distributions, including Lévy-stable, GED, Student-t, and others. The FELW estimator remains consistent and resilient, thereby positioning itself as a reliable benchmark for evaluating the performance of ML models in long-memory environments.

Our investigation reveals that increased model complexity does not guarantee superior predictive performance, although hybrid architectures may outperform individual models under certain conditions. Notably, DNN demonstrate significant improvements in capturing long memory, particularly when architectural depth is increased. Among all models considered, the Attention-based LSTM exhibits superior capability in preserving memory features and adapting to regime environments, making it a strong candidate for both predictive tasks and market efficiency estimation.

We propose two tests that explain the divergent performance of machine learning models by assessing their ability to preserve long memory. These tests provide novel insights into model selection for directional prediction, particularly when compared to the traditional MSE metric and evaluated under complex, inefficient market dynamics. As directional prediction has substantial practical relevance, our tests offer important guidance for developing more effective ML models for trading strategies.

Based on the Type I test, we find that seven classical ML approaches have very poor performance in modeling long memory, explaining why these models are not robust and have weak ability of generalization. In contrast, Seq2Seq models are capable of maintaining long-range dependence accurately, providing interpretability from a long-memory perspective. Specifically, LSTM, Attention-LSTM and Embedding-Attention-LSTM models can be the potential first-choice models for financial long-memory time-series prediction.

In addition, this study is the first to assess the effectiveness of Seq2Seq models in capturing fractionally cointegrated series empirically, providing new insights into these models' capacity to handle complex time dependence in financial markets. This not only fosters the development of more advanced models, but also promotes the generalization of these models across diverse financial datasets, making a notable contribution to the ongoing discussion on ML model validation and applicability. By filtering out models that are ineffective in processing long-memory data, the tests can enhance the efficiency of model selection, optimizing the modeling process for financial applications. These findings are critical for refining model specifications to better leverage long-memory characteristics, ultimately helping to create more robust and effective trading strategies.

Our methodology can be readily generalized to form a universal framework for evaluating the ability of ML models learning other statistical properties, highlighting the importance of statistical consistency, filling a significant gap in the financial markets literature. This approach bridges the divide between traditional statistical techniques and ML, emphasizing the importance of incorporating statistical properties, such as long memory, into ML models to ensure both robustness and generalization as well as to provide interpretability. Our findings underscore the need for a systematic approach to model selection and validation in complex financial markets, offering researchers and practitioners a valuable tool to enhance profitability and reduce risk through more informed decision-making. By integrating statistical methods with ML, our research sheds new light on future advancements in the field.

CRediT authorship contribution statement

Shuyue Li: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Larisa Yarovaya:** Writing – review & editing, Writing – original draft, Validation, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Tapas Mishra:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Investigation, Formal analysis, Conceptualization.

Appendix

Hill Estimator for Heavy-Tailed Distributions

The Hill estimator by Hill (1975) is a frequently cited method in extreme value theory designed to measure the thickness of a distribution's upper tail. It presumes that the tail of the underlying distribution exhibits behavior similar to a Pareto-type function. Specifically, one focuses on the k largest observations within a sample of size n, denoted by $X_{(1)} \geq X_{(2)} \geq \cdots \geq X_{(n)}$, where $X_{(i)}$ represents the ith order statistic. The Hill estimator for the tail index is given by

$$\hat{\alpha}_H(k) = \left[\frac{1}{k} \sum_{i=1}^{k} \left(\ln X_{(i)} - \ln X_{(k+1)} \right) \right]^{-1}.$$

A suitable choice of k is essential to achieve a balance between bias and variance: including more top-order statistics (k large) can lower variance but increase potential bias if the assumed Pareto behavior does not hold for lesser values, while using fewer observations (k small) limits bias at the expense of higher variance. By judiciously selecting k, the Hill estimator offers a practical means to capture and quantify the tail thickness of distributions in various financial and economic applications.

Impact of Long Memory to Various Distributions

Data Generation Process

This study systematically generates 12 synthetic datasets based on distinct probability distributions commonly employed in financial econometrics and risk management. The objective is to create controlled experimental data for the subsequent evaluation of statistical and econometric models. The datasets, each comprising 500×500 observations, are generated using the `Python` programming language, leveraging the `NumPy` and `SciPy` libraries. To ensure reproducibility of the stochastic processes, the random number generator is initialized with a fixed seed (`np.random.seed(42)`). The details of the data generation process are as follows. For each distribution, figures (a) to (f) illustrate: (a) the original data sample, (b) fractionally differenced sequences for different

d values, (c) cumulative sums of those fractionally differenced data, (d) average cumulative sums, (e) heavy tail ratios for the fractionally differenced sequences, and (f) heavy tail ratios for the cumulative sum sequences. Figs. 22(a) to 33(a) show the original samples for each of the twelve distributions. Heavy tailed distributions, such as Levy Stable, Generalized Hyperbolic, Student t, and Skewed t, exhibit substantial tail thickness in the raw data. Light or moderately tailed distributions, such as Normal, NIG, and GED, show fewer extreme observations in their original samples.

When fractional differencing is applied with parameter $d > 0$, as in Figs. 22(b) to 33(b), the differenced data often display reduced persistence and altered tail behavior. For heavier tailed distributions like Levy Stable or Student t, increasing d can diminish or smooth large swings, though tail thickness may still be partly observed. For comparatively light tailed distributions such as Normal or GED, changes in tail thickness are subtle, and the differenced series appear closer to weakly dependent or near white noise structures. Figs. 22(c) to 33(c) show the cumulative sums of the fractionally differenced data. Cumulative summation partially integrates these increments, reintroducing some dependence. In originally heavy tailed distributions, cumulative sums can retain sizeable fluctuations, though the net effect of differencing is to lessen extreme outcomes. In light tailed distributions, cumulative sums exhibit more regular behavior, since large deviations occur less frequently.

Figs. 22(d) to 33(d) present the average cumulative sum of the fractionally differenced sequences for each d. When these differenced increments are cumulatively summed, certain memory properties and long-range dependence can be partially reintroduced. For larger d, the cumulative sum tends to restore some of the dynamics seen in the original series, emphasizing slow-moving or drifting behavior over time. In contrast, for smaller d, the cumulative sum may exhibit less pronounced drift but still reveals enhanced temporal dependence compared with the purely differenced data.

Subplots (e) and (f) of Figs. 22 to 33 illustrate heavy tail ratios for the fractionally differenced data and for their cumulative sums, respectively. The heavy tail ratio measures the proportion of observations crossing a high threshold, thereby indicating how tail risk or extreme values evolve under each transformation. From a memory enhancement viewpoint, the cumulative sum frequently raises this ratio relative to the purely differenced data, since summing the increments can amplify rare but large shocks.

For heavier tailed distributions such as Levy Stable, Generalized Hyperbolic, and Student t, Subplot (e) shows that fractional differencing reduces the frequency of extreme observations, lowering the heavy tail ratio compared with the original $d = 0$ sample. However, Subplot (f) reveals that the cumulative sum can restore or even magnify heavy tail features, reflecting how partial reintegration reintroduces some of the original memory and increases the likelihood of large aggregated values.

For moderately or light tailed distributions such as Normal, Extreme Value, and GED, Subplot (e) indicates that fractional differencing alone causes minimal changes in the heavy tail ratio, given their thinner tails. Meanwhile, Subplot (f) shows that the cumulative sum may still slightly elevate the tail ratio compared with the differenced data, though the effect is weaker than in heavier tailed cases.

Overall, fractional differencing suppresses extreme observations, but cumulatively summing the differenced data can restore dependence and intensify tail behavior. From a memory enhancement perspective, this result underscores the interplay between differencing in shaping both the persistence and the tail thickness of the resulting series.

1. **Lévy Stable Distribution**
   A Lévy α-stable distribution is generated with parameters $\alpha = 1.7$ and $\beta = 0$:
   $$X_{i,j} \sim S(\alpha = 1.7, \beta = 0), \quad i, j = 1, \ldots, 500. \tag{70}$$
   Note: α $(0 < \alpha \leq 2)$ is the characteristic exponent controlling the tail thickness (smaller α implies heavier tails), while β $(-1 \leq \beta \leq 1)$ governs the skewness ($\beta = 0$ indicates symmetry). Additional location and scale parameters can also be included but are taken as default here.
2. **Generalized Error Distribution (GED)**
   The generalized normal (error) distribution is simulated using $\beta = 1.5$, which controls the kurtosis:
   $$X_{i,j} \sim \text{GED}(\beta = 1.5), \quad i, j = 1, \ldots, 500. \tag{71}$$
   Note: β determines the tail behavior and peakedness of the distribution. A larger β tends to yield thinner tails compared to the normal distribution, whereas a smaller β produces heavier tails.
3. **Asymmetric Power-Law Distribution**
   Based on the Pareto distribution with shape parameter $\alpha = 2.5$, asymmetry is introduced by assigning sign probabilities:
   $$X = (P + 1) \cdot S, \tag{72}$$
   $$S = \begin{cases} +1, & \text{with probability } 0.5 + \frac{\text{skew}}{2}, \\ -1, & \text{otherwise}, \end{cases} \tag{73}$$
   where $P \sim \text{Pareto}(\alpha = 2.5)$ and skew $= 0.3$.
   Note: α is the Pareto shape parameter (heavier tails occur with smaller α). The skew value controls the probability of generating positive versus negative values, introducing asymmetry.
4. **Generalized Extreme Value (GEV) Distribution**
   Observations are drawn from the GEV distribution with shape parameter $c = 0.1$:
   $$X_{i,j} \sim \text{GEV}(c = 0.1), \quad i, j = 1, \ldots, 500. \tag{74}$$
   Note: The shape parameter c dictates whether the distribution corresponds to the Gumbel ($c \to 0$), Fréchet ($c > 0$), or Weibull ($c < 0$) type, thus controlling the thickness of the tail.

5. **Generalized Hyperbolic (GH) Distribution**
Samples are generated from the generalized hyperbolic distribution with parameters $a = 1$, $b = 1$, $p = 1$, and $q = 1$:

$$X_{i,j} \sim \mathrm{GH}(a = 1, b = 1, p = 1, q = 1), \quad i, j = 1, \ldots, 500. \tag{75}$$

Note: In the GH family, a and b primarily control the tail heaviness and skewness, while p and q adjust the shape of the distribution's body and tails in more nuanced ways.

6. **Student's t-Distribution**
Data are generated from a Student's t-distribution with $\nu = 5$ degrees of freedom:

$$X_{i,j} \sim t_5, \quad i, j = 1, \ldots, 500. \tag{76}$$

Note: ν (degrees of freedom) controls tail thickness. As ν increases, the Student's t-distribution approaches the normal distribution. Smaller ν corresponds to heavier tails.

7. **Skewed t-Distribution**
A skewed t-distribution with $\nu = 5$ and skewness parameter $\delta = 0.5$ is simulated. The stochastic representation is given by:

$$X = \frac{\delta |U| + \sqrt{1 - \delta^2}\, Z}{\sqrt{V/\nu}}, \tag{77}$$

where $U, Z \sim \mathcal{N}(0, 1)$ and $V \sim \chi^2_\nu$.
Note: ν again controls the tail thickness (as in the symmetric Student's t), while δ induces skewness by asymmetrically scaling the positive side relative to the negative side.

8. **Generalized Lambda Distribution (GLD)**
The RS-parameterized quantile function is employed to simulate data from the GLD:

$$X = \lambda_1 + \frac{U^{\lambda_3} - (1 - U)^{\lambda_4}}{\lambda_2}, \quad U \sim \mathcal{U}(0, 1), \tag{78}$$

with parameters $\lambda_1 = 0$, $\lambda_2 = 1$, $\lambda_3 = 0.1$, $\lambda_4 = 0.1$.
Note: λ_1 and λ_2 control location and scale, while λ_3 and λ_4 govern the shape and tail behavior, allowing the GLD to approximate a wide range of distributions.

9. **Generalized Pareto (GPD) Distribution**
Observations are drawn from the GPD with shape parameter $c = 0.1$:

$$X_{i,j} \sim \mathrm{GPD}(c = 0.1), \quad i, j = 1, \ldots, 500. \tag{79}$$

Note: The shape parameter c (ξ in many texts) controls the thickness of the right tail. Positive values produce heavy-tailed distributions, whereas negative values yield bounded upper tails.

10. **Normal Distribution (Gaussian)**
A standard normal distribution with zero mean and unit variance is simulated:

$$X_{i,j} \sim \mathcal{N}(0, 1), \quad i, j = 1, \ldots, 500. \tag{80}$$

Note: The normal distribution is fully characterized by its mean (0) and variance (1). It is symmetric and exhibits relatively light tails compared to many other distributions used in finance.

11. **Normal Inverse Gaussian (NIG) Distribution**
Observations are drawn from the NIG distribution with parameters $a = 2$ and $b = 1$, ensuring $|b| < a$ for validity:

$$X_{i,j} \sim \mathrm{NIG}(a = 2, b = 1), \quad i, j = 1, \ldots, 500. \tag{81}$$

Note: In the NIG distribution, a (the tail or steepness parameter) and b (the asymmetry parameter) together shape both the skewness and kurtosis. The conditions $|b| < a$ ensure a well-defined distribution.

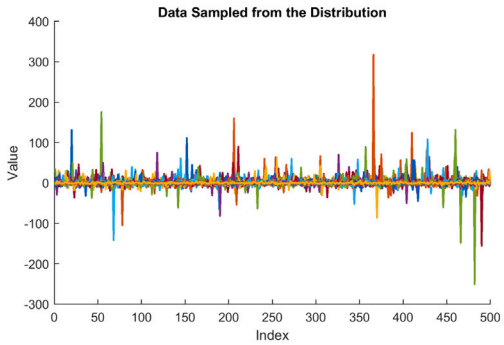
12. **Skewed Generalized Error Distribution**
A skewed version of the GED is generated by asymmetrically scaling the positive and negative values:

$$X = \begin{cases} x(1 + \mathrm{skew}), & x \geq 0, \\ x(1 - \mathrm{skew}), & x < 0, \end{cases} \tag{82}$$

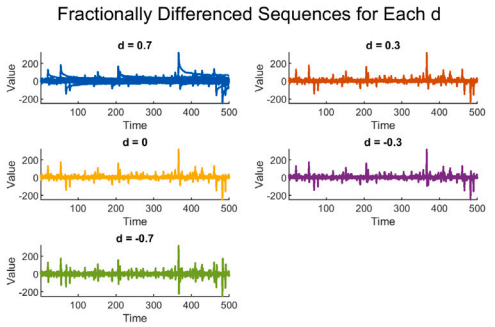
where $x \sim \mathrm{GED}(\beta = 1.5)$ and $\mathrm{skew} = 0.3$.
Note: Here, β (as in the standard GED) controls the kurtosis and tail behavior, while skew introduces asymmetry by scaling the magnitudes of positive and negative values differently.

Figures of various distributions

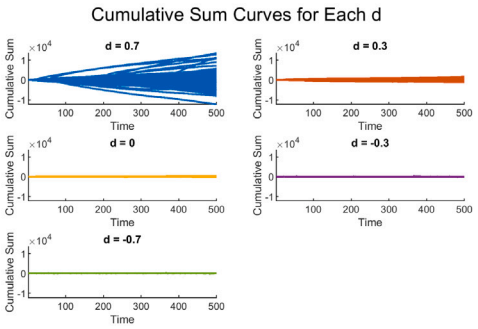
See Figs. 22–33.



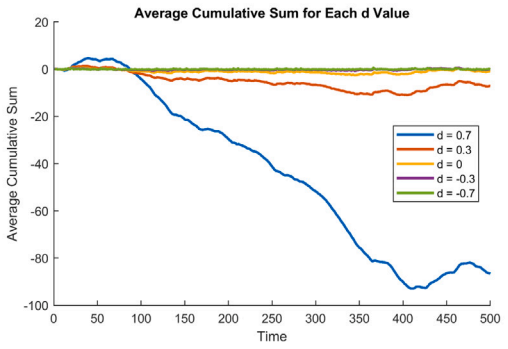
(a) Data Sample from the Levy Stable Distribution



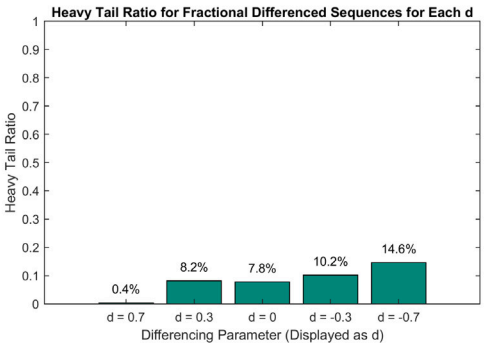
(b) Fractionally Differenced Sequences for Each d



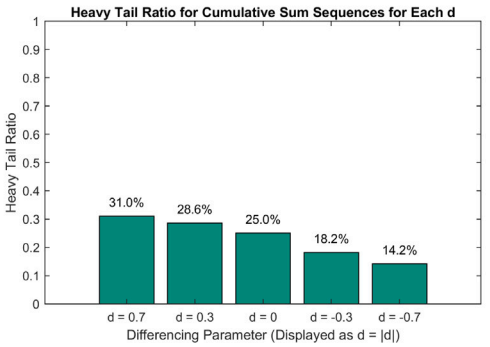
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

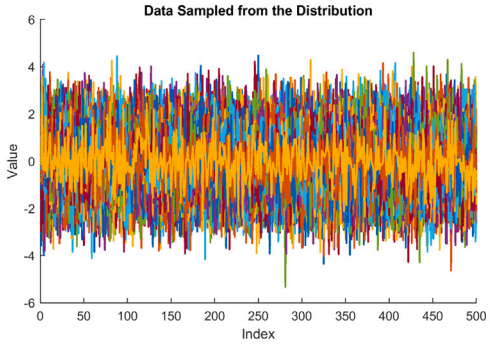


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

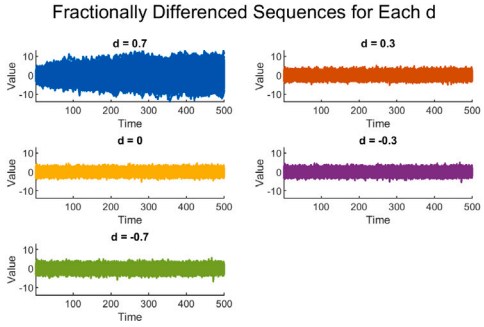


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

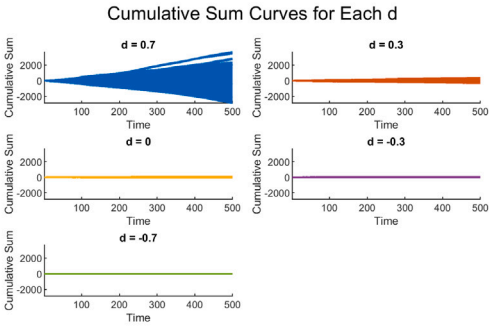
Fig. 22. Levy `Stable` distribution.



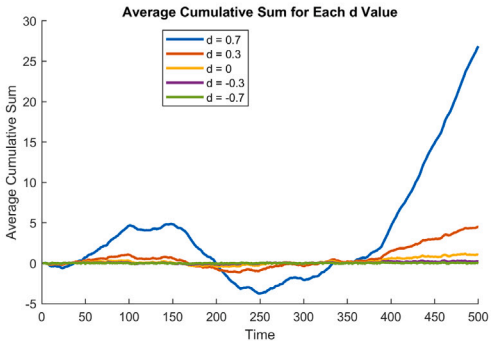
(a) Data Sample from the GED Distribution



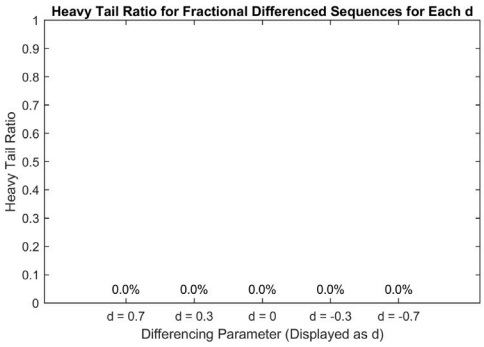
(b) Fractionally Differenced Sequences for Each d



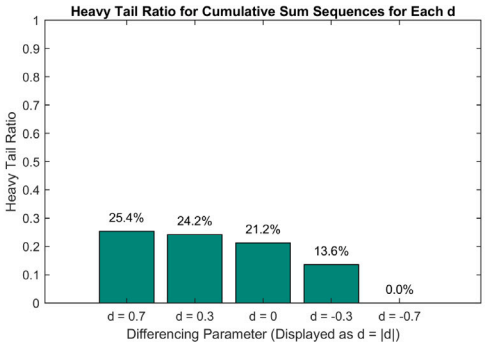
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

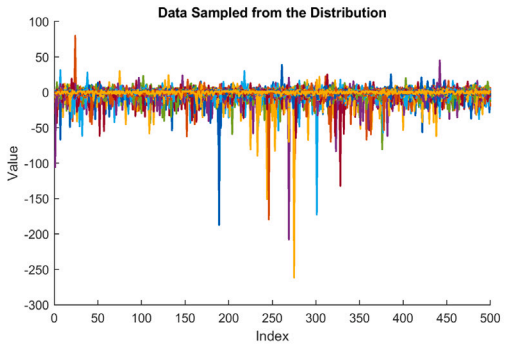


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

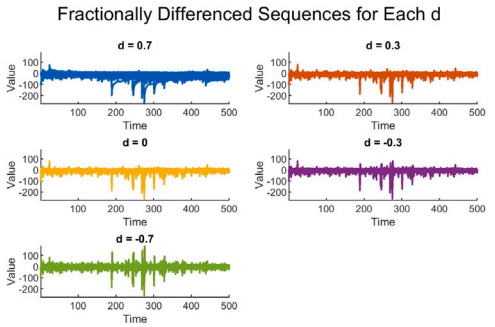


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

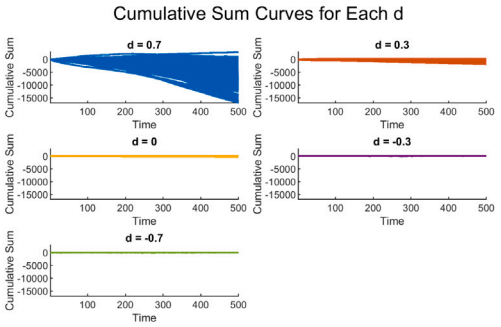
Fig. 23. Ged distribution.



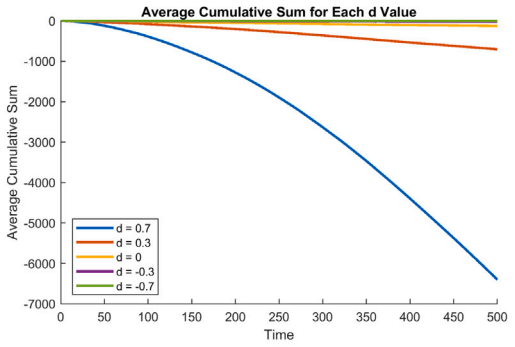
(a) Data Sample from the Asymmetric Power Distribution



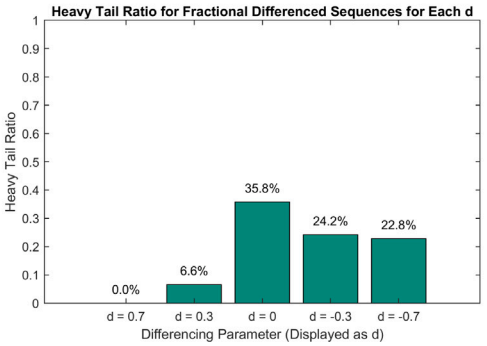
(b) Fractionally Differenced Sequences for Each d



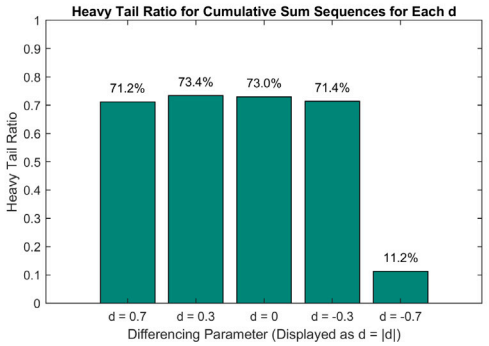
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

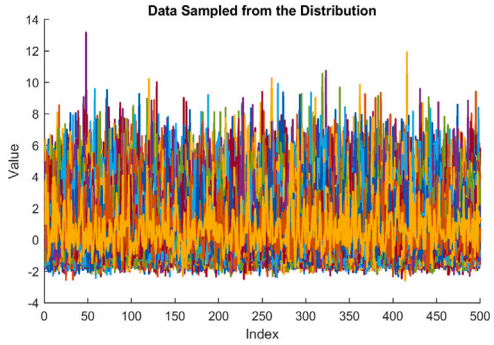


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

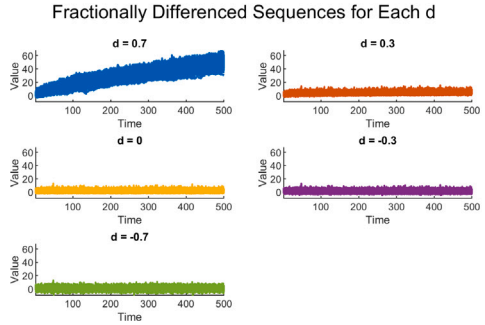


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

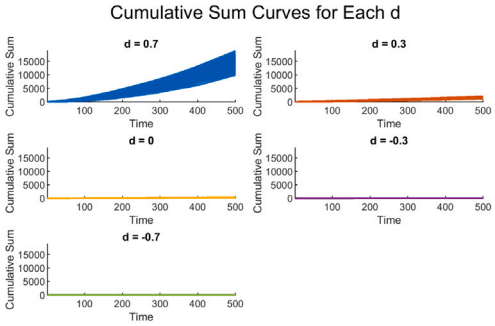
Fig. 24. `Asymmetric Power` distribution.



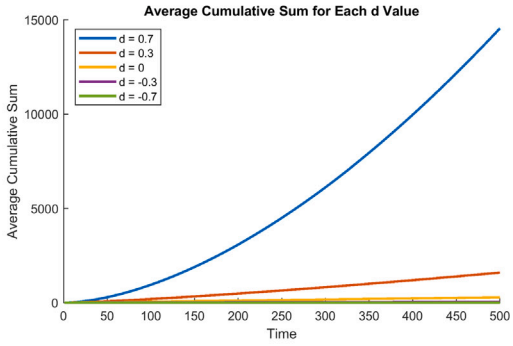
(a) Data Sample from the Extreme Value Distribution



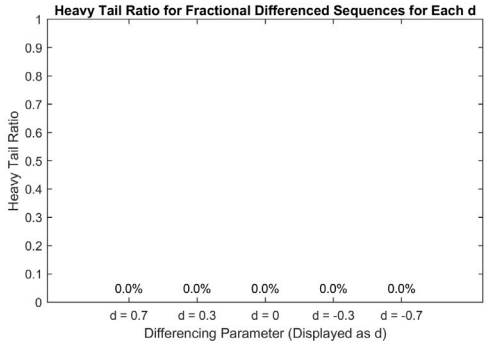
(b) Fractionally Differenced Sequences for Each d



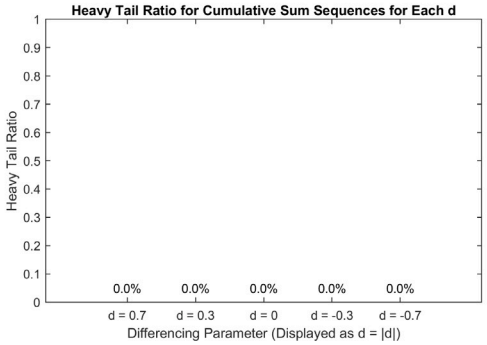
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

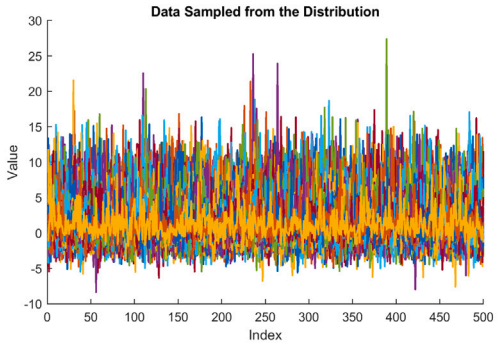


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

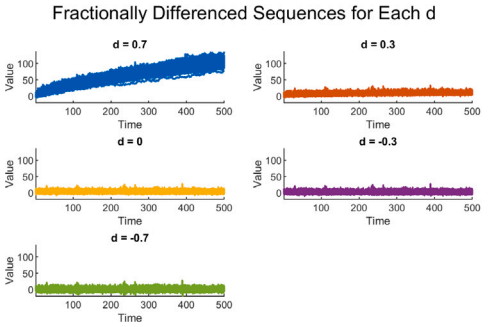


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

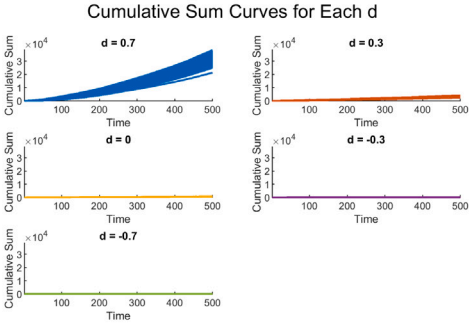
Fig. 25. `Extreme Value` distribution.



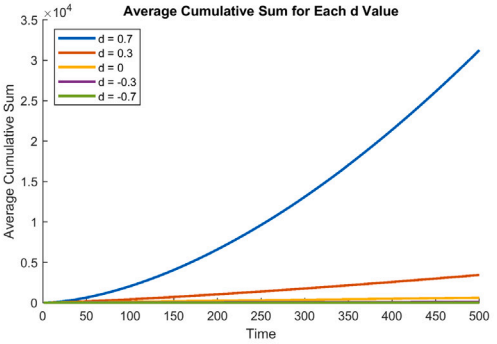
(a) Data Sample from the Generalized Hyperbolic Distribution



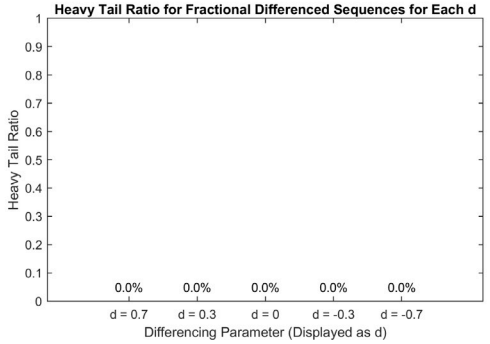
(b) Fractionally Differenced Sequences for Each d



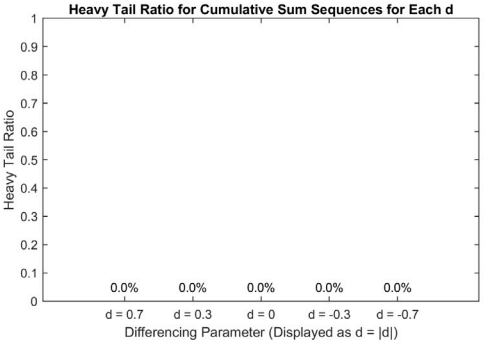
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

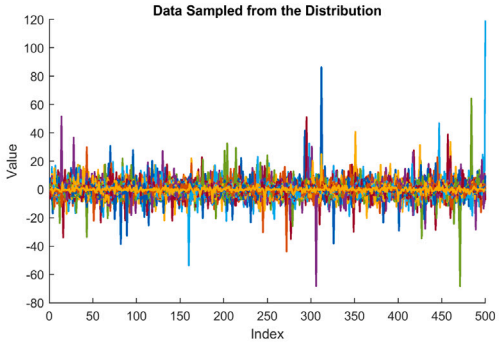


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

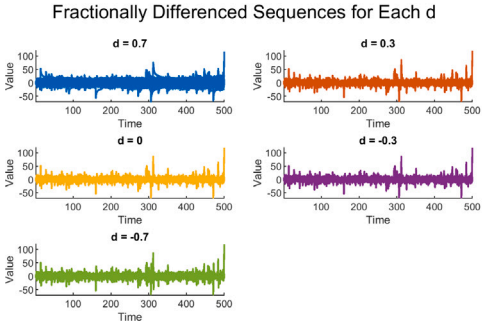


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

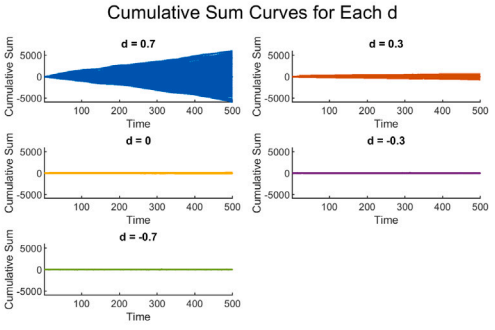
Fig. 26. `Generalized Hyperbolic` distribution.



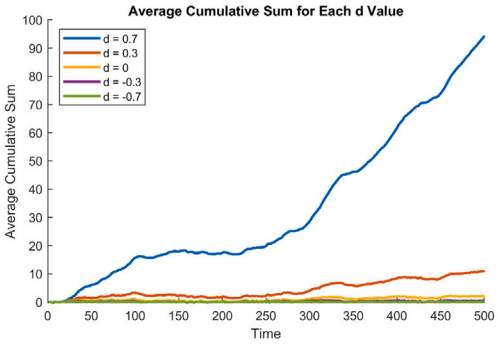
(a) Data Sample from the student t Distribution



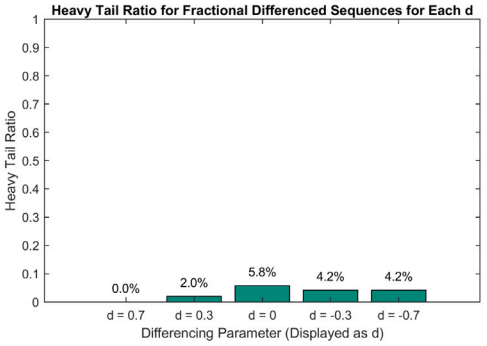
(b) Fractionally Differenced Sequences for Each d



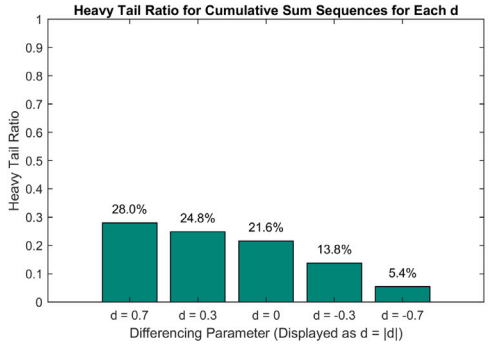
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

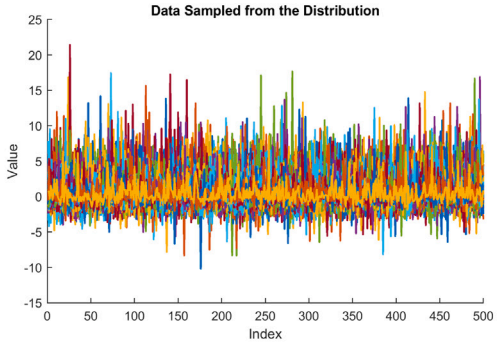


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

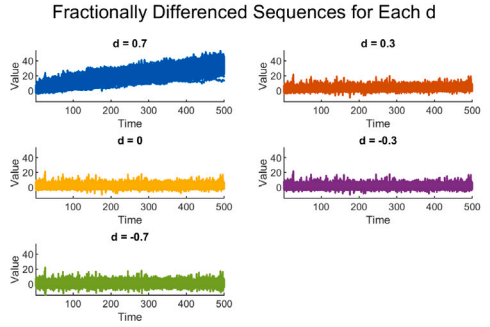


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

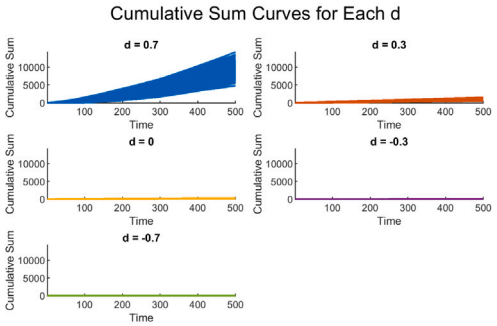
Fig. 27. `Student T` distribution.



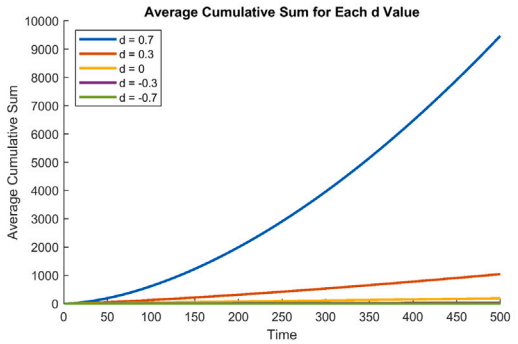
(a) Data Sample from the Skewed t Distribution



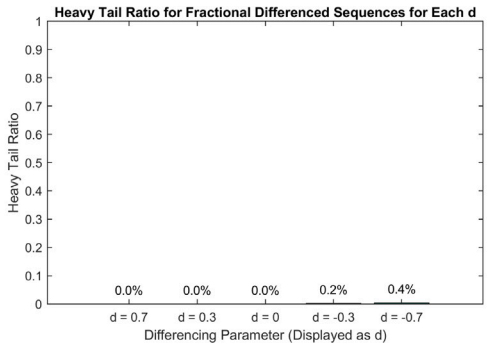
(b) Fractionally Differenced Sequences for Each d



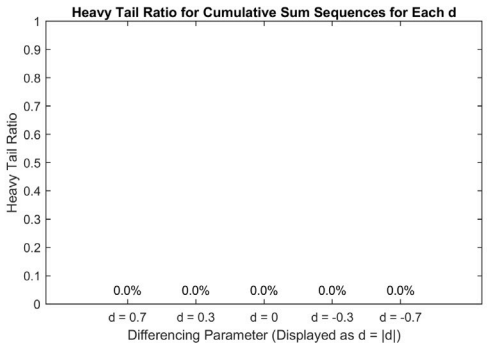
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

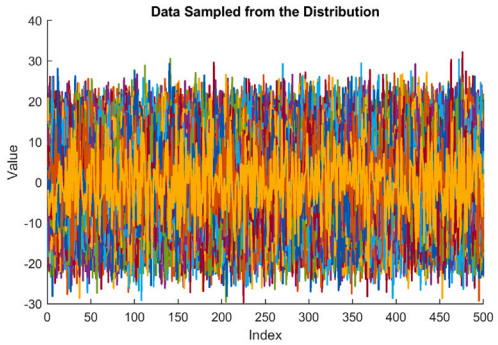


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

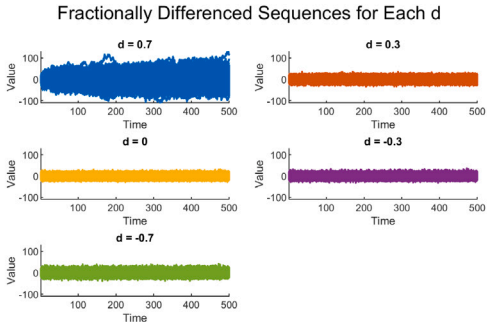


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

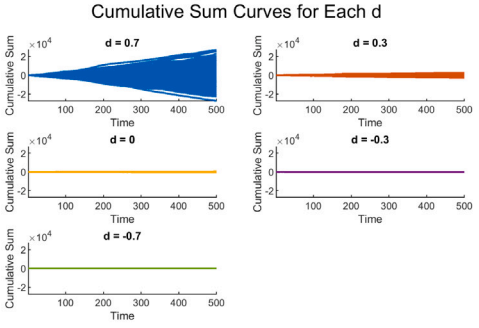
Fig. 28. Skewed T distribution.



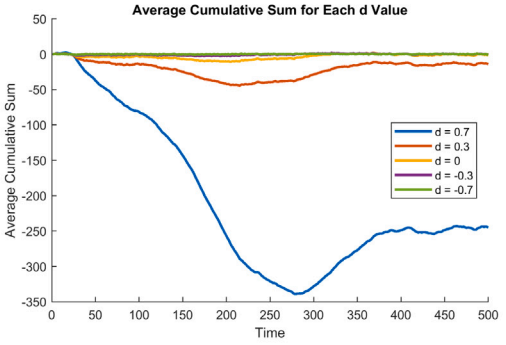
(a) Data Sample from the Generalized Lambda Distribution



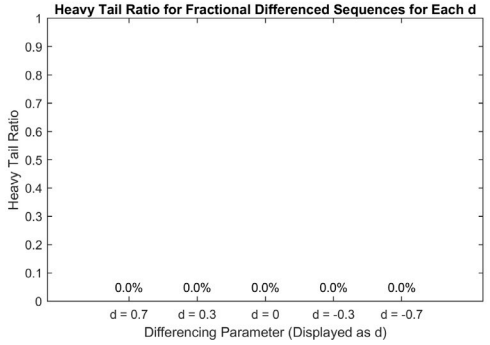
(b) Fractionally Differenced Sequences for Each d



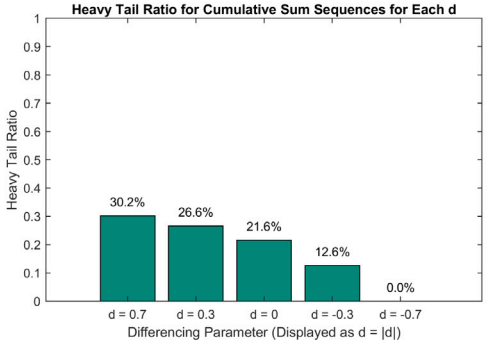
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

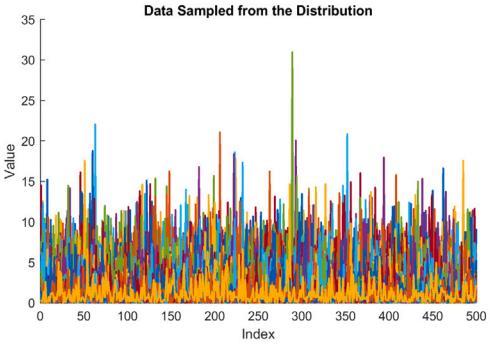


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

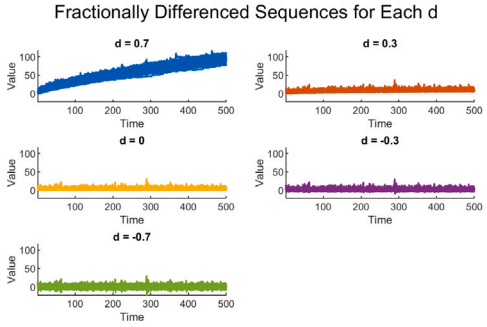


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

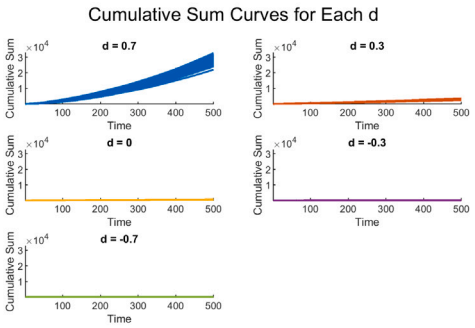
Fig. 29. `Generalized Lambda` distribution.



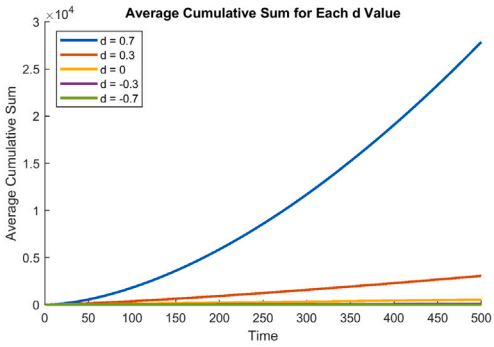
(a) Data Sample from the Generalized Pareto Distribution



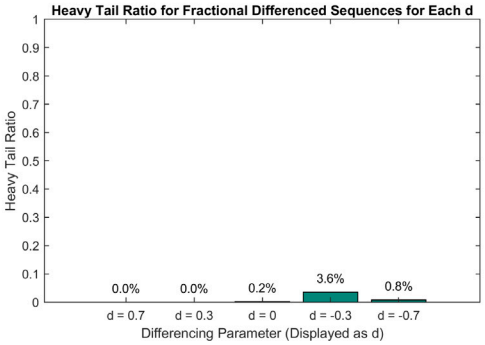
(b) Fractionally Differenced Sequences for Each d



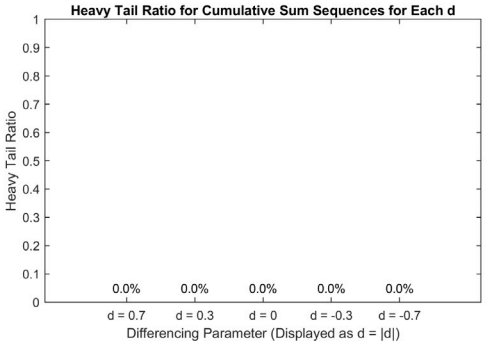
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

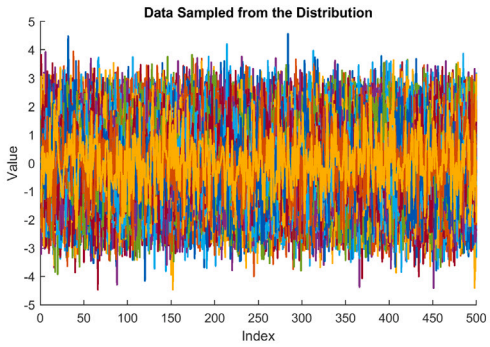


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

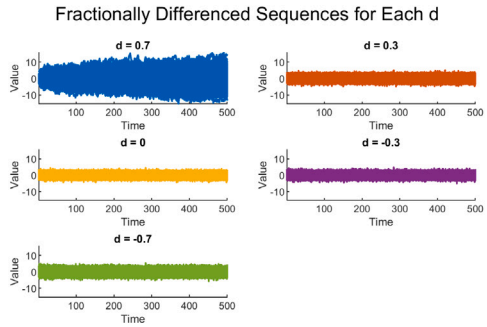


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

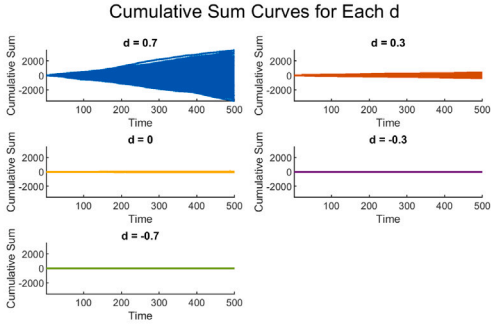
Fig. 30. `Generalized Pareto` distribution.



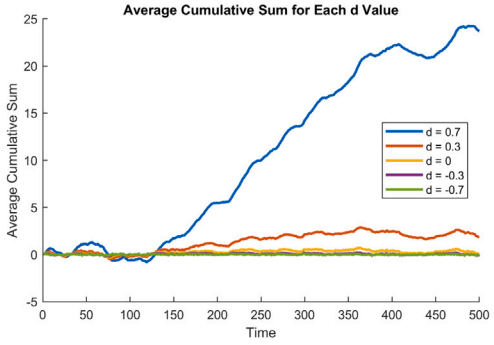
(a) Data Sample from the Normal Distribution



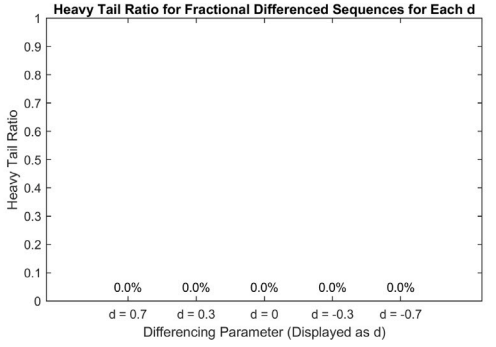
(b) Fractionally Differenced Sequences for Each d



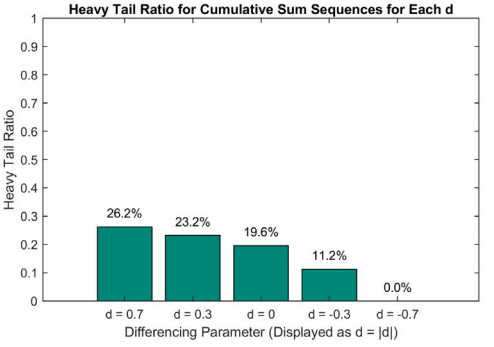
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

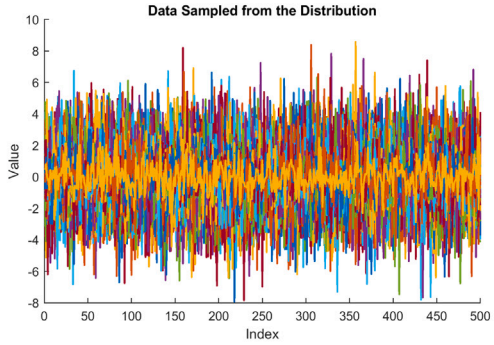


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

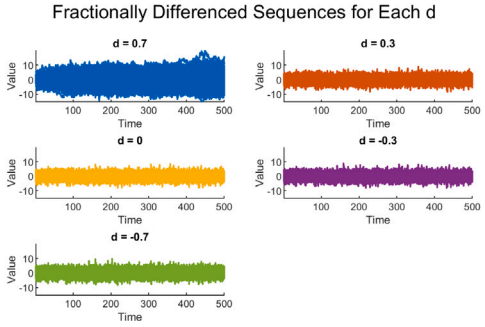


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

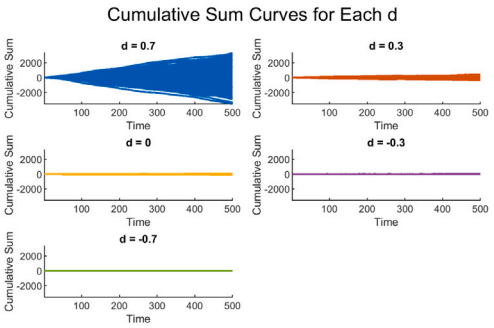
Fig. 31. `Normal` distribution.



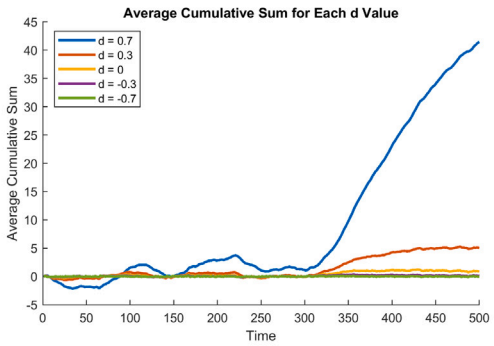
(a) Data Sample from the Normal Inverse Gaussian Distribution



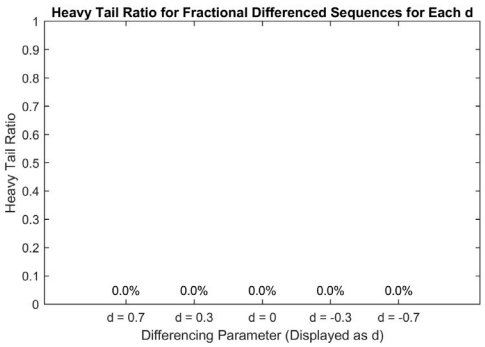
(b) Fractionally Differenced Sequences for Each d



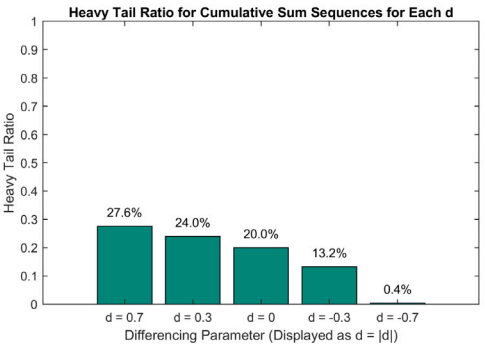
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value

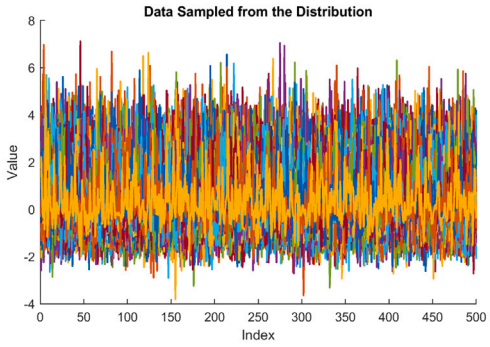


(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d

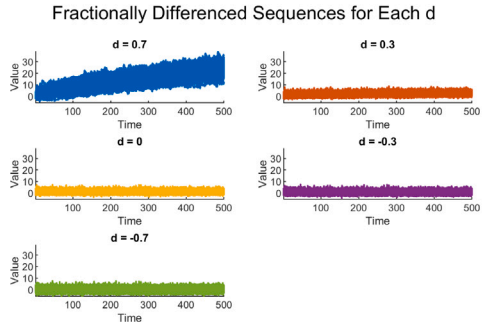


(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

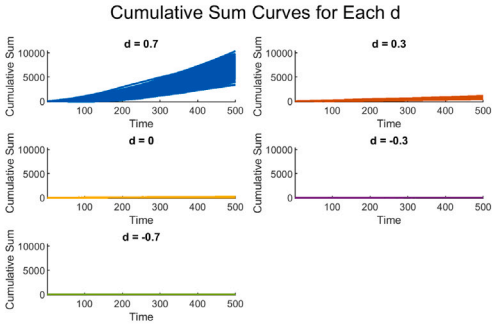
Fig. 32. `Normal Inverse Gaussian` distribution.



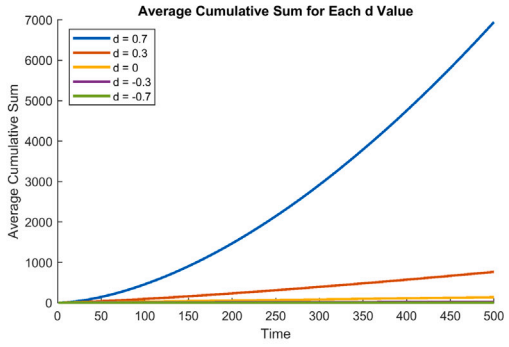
(a) Data Sample from the Skewed Ged Distribution



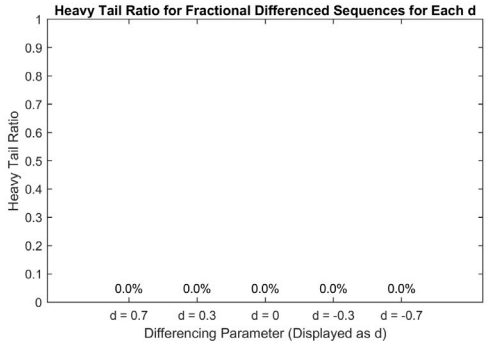
(b) Fractionally Differenced Sequences for Each d



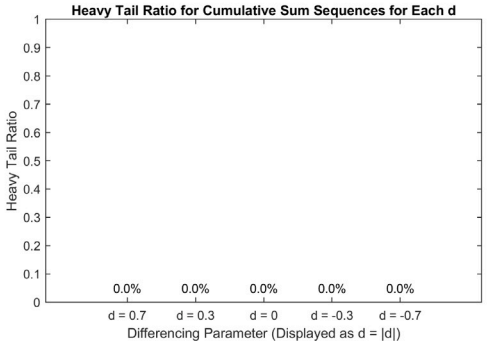
(c) Cumulative Sum Curves for Each d



(d) Average Cumulative Sum for Each d Value



(e) Heavy Tail Ratio for Fractionally Differenced Sequences for Each d



(f) Heavy Tail Ratio for Cumulative Sum Sequences for Each d

Fig. 33. Skewed GED distribution.

Data availability

Data will be made available on request.

References

Ahmed, N.K., Atiya, A.F., Gayar, N.E., El-Shishiny, H., 2010. An empirical comparison of machine learning models for time series forecasting. Econometric Rev. 29 (5–6), 594–621.

Alexandridis, A.K., Panopoulou, E., Souropanis, I., 2024. Forecasting exchange rate volatility: An amalgamation approach. J. Int. Financ. Mark. Inst. Money 97, 102067.

Alsharef, A., Aggarwal, K., Sonia, Kumar, M., Mishra, A., 2022a. Review of ML and AutoML solutions to forecast time-series data. Arch. Comput. Methods Eng. 29 (7), 5297–5311.
Alsharef, A., Kumar, K., Iwendi, C., 2022b. Time series data modeling using advanced machine learning and AutoML. Sustainability 14 (22), 15292.
Amadeo, A.J., Siento, J.G., Eikwine, T.A., Parmonangan, I.H., et al., 2023. Temporal fusion transformer for multi horizon bitcoin price forecasting. In: 2023 IEEE 9th Information Technology International Seminar. ITIS, IEEE, pp. 1–7.
Amirshahi, B., Lahmiri, S., 2023. Hybrid deep learning and GARCH-family models for forecasting volatility of cryptocurrencies. Mach. Learn. Appl. 12, 100465.
Ampountolas, A., 2024. Enhancing forecasting accuracy in commodity and financial markets: Insights from garch and svr models. Int. J. Financ. Stud. 12 (3), 59.
Ardia, D., Bluteau, K., Rüede, M., 2019. Regime changes in bitcoin GARCH volatility dynamics. Financ. Res. Lett. 29, 266–271.
Aygun, B., Gunay, E.K., 2021. Comparison of statistical and machine learning algorithms for forecasting daily bitcoin returns. Avrupa Bilim. Ve Teknol. Derg. (21), 444–454.
Baruník, J., Dvořáková, S., 2015. An empirical model of fractionally cointegrated daily high and low stock market prices. Econ. Model. 45, 193–206.
Begušić, S., Kostanjčar, Z., Stanley, H.E., Podobnik, B., 2018. Scaling properties of extreme price fluctuations in bitcoin markets. Phys. A 510, 400–406.
Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. 5 (2), 157–166.
Boros, D., Csanády, B., Ivkovic, I., Nagy, L., Lukács, A., Márkus, L., 2024. Deep learning the hurst parameter of linear fractional processes and assessing its reliability. arXiv preprint arXiv:2401.01789.
Bouri, E., Christou, C., Gupta, R., 2022. Forecasting returns of major cryptocurrencies: Evidence from regime-switching factor models. Financ. Res. Lett. 49, 103193.
Breiman, L., 2001. Random forests. Mach. Learn. 45, 5–32.
Bundi, N., Wildi, M., 2019. Bitcoin and market-(in) efficiency: a systematic time series approach. Digit. Financ. 1 (1), 47–65.
Buthelezi, E.M., 2024. Navigating global uncertainty: Examining the effect of geopolitical risks on cryptocurrency prices and volatility in a Markov-switching vector autoregressive model. Int. Econ. J. 38 (4), 564–590.
Caporale, G.M., Gil-Alana, L., Plastun, A., 2018. Persistence in the cryptocurrency market. Res. Int. Bus. Financ. 46, 141–148.
Cerqueira, V., Torgo, L., Soares, C., 2022. A case study comparing machine learning with statistical methods for time series forecasting: size matters. J. Intell. Inf. Syst. 59 (2), 415–433.
Chalkiadakis, I., Peters, G.W., Ames, M., 2023. Hybrid ARDL-MIDAS-transformer time-series regressions for multi-topic crypto market sentiment driven by price and technology factors. Digit. Financ. 5 (2), 295–365.
Chatterjee, A., Bhowmick, H., Sen, J., 2021. Stock price prediction using time series, econometric, machine learning, and deep learning models. In: 2021 IEEE Mysore Sub Section International Conference. MysuruCon, IEEE, pp. 289–296.
Chatterjee, A., Bhowmick, H., Sen, J., 2022. Stock volatility prediction using time series and deep learning approach. In: 2022 IEEE 2nd Mysore Sub Section International Conference. MysuruCon, IEEE, pp. 1–6.
Cheah, E.-T., Fry, J., 2015. Speculative bubbles in bitcoin markets? An empirical investigation into the fundamental value of bitcoin. Econom. Lett. 130, 32–36.
Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
Ciaburro, G., Iannace, G., 2021. Machine learning-based algorithms to knowledge extraction from time series data: A review. Data 6 (6), 55.
Ciaian, P., Rajcaniova, M., et al., 2018. Virtual relationships: Short-and long-run evidence from BitCoin and altcoin markets. J. Int. Financ. Mark. Institutions Money 52, 173–195.
Corbet, S., Lucey, B., Urquhart, A., Yarovaya, L., 2019. Cryptocurrencies as a financial asset: A systematic analysis. Int. Rev. Financ. Anal. 62, 182–199.
Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20, 273–297.
D'Ecclesia, R.L., Clementi, D., 2021. Volatility in the stock market: ANN versus parametric models. Ann. Oper. Res. 299 (1), 1101–1127.
Derbentsev, V., Datsenko, N., Stepanenko, O., Bezkorovainyi, V., 2019. Forecasting cryptocurrency prices time series using machine learning approach. In: SHS Web of Conferences, vol. 65, EDP Sciences, p. 02001.
Derbentsev, V., Matviychuk, A., Datsenko, N., Bezkorovainyi, V., Azaryan, A., 2020. Machine Learning Approaches for Financial Time Series Forecasting. CEUR Workshop Proceedings.
Dingli, A., Fournier, K.S., 2017. Financial time series forecasting-a deep learning approach. Int. J. Mach. Learn. Comput. 7 (5), 118–122.
Diniz-Maganini, N., Rasheed, A.A., Sheng, H.H., 2021. Exchange rate regimes and price efficiency: Empirical examination of the impact of financial crisis. J. Int. Financ. Mark. Inst. Money 73, 101361.
Drucker, H., Burges, C.J., Kaufman, L., Smola, A., Vapnik, V., 1996. Support vector regression machines. Adv. Neural Inf. Process. Syst. 9.
Du Bois, N., Hollywood, L., Coyle, D., et al., 2023. State-of-the-art deep learning models are superior for time series forecasting and are applied optimally with iterative prediction methods.
Duan, K., Gao, Y., Mishra, T., Satchell, S., 2023. Efficiency dynamics across segmented bitcoin markets: Evidence from a decomposition strategy. J. Int. Financ. Mark. Inst. Money 83, 101742.
Duan, K., Li, Z., Urquhart, A., Ye, J., 2021. Dynamic efficiency and arbitrage potential in bitcoin: A long-memory approach. Int. Rev. Financ. Anal. 75, 101725.
Fama, E.F., 1970. Efficient capital markets. J. Financ. 25 (2), 383–417.
Fang, W., 2023. Transformer and long short-term memory networks for long sequence time sequence forecasting problem. CISAI 2022, In: Fifth International Conference on Computer Information Science and Artificial Intelligence, vol. 12566, SPIE, pp. 228–234.
Fischer, T., Krauss, C., Treichel, A., 2018. Machine learning for time series forecasting-a simulation study. Technical Report, FAU Discussion Papers in Economics.
García-Medina, A., Aguayo-Moreno, E., 2024. LSTM–GARCH hybrid model for the prediction of volatility in cryptocurrency portfolios. Comput. Econ. 63 (4), 1511–1542.
Gezici, B., Sefer, E., 2024. Deep transformer-based asset price and direction prediction. IEEE Access.
Graves, A., Schmidhuber, J., 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. 18 (5–6), 602–610.
Grobys, K., 2023. A multifractal model of asset (in) variances. J. Int. Financ. Mark. Inst. Money 85, 101767.
Grobys, K., Dufitinema, J., Sapkota, N., Kolari, J.W., 2022. What's the expected loss when bitcoin is under cyberattack? A fractal process analysis. J. Int. Financ. Mark. Inst. Money 77, 101534.
Hassler, U., 2018. Time Series Analysis with Long Memory in View. John Wiley & Sons.
Hill, B.M., 1975. A simple general approach to inference about the tail of a distribution. Ann. Statist. 1163–1174.
Hirano, Y., Pichl, L., Eom, C., Kaizoji, T., 2018. Analysis of bitcoin market efficiency by using machine learning. In: CBU International Conference Proceedings, vol. 6, pp. 175–180.
Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.
Hoerl, A.E., Kennard, R.W., 1970. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 12 (1), 55–67.
Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Netw. 2 (5), 359–366.
Hossain, A., Nasser, M., 2008. Comparison of GARCH and neural network methods in financial time series prediction. In: 2008 11th International Conference on Computer and Information Technology. IEEE, pp. 729–734.

Hotta, L., Trucios, C., Pereira, P.L.V., Zevallos, M., 2025. Forecasting bitcoin and ethereum risk measures through MSGARCH models: Does the specification matter? Braz. Rev. Financ. 23, e202503–e202503.
Hsu, M.-W., Lessmann, S., Sung, M.-C., Ma, T., Johnson, J.E., 2016. Bridging the divide in financial market forecasting: machine learners vs. financial economists. Expert Syst. Appl. 61, 215–234.
Hu, Y., Valera, H.G.A., Oxley, L., 2019. Market efficiency of the top market-cap cryptocurrencies: Further evidence from a panel framework. Financ. Res. Lett. 31, 138–145.
Huang, Y., Duan, K., Urquhart, A., 2023. Time-varying dependence between bitcoin and green financial assets: A comparison between pre-and post-COVID-19 periods. J. Int. Financ. Mark. Inst. Money 82, 101687.
Huang, Z.-C., Sangiorgi, I., Urquhart, A., 2024. Forecasting bitcoin volatility using machine learning techniques. J. Int. Financ. Mark. Inst. Money 97, 102064.
Júnior, D.S.d.O.S., de Oliveira, J.F., de Mattos Neto, P.S., 2019. An intelligent hybridization of ARIMA with machine learning models for time series forecasting. Knowl.-Based Syst. 175, 72–86.
Kakinaka, S., Umeno, K., 2020. Characterizing cryptocurrency market with Lévy's stable distributions. J. Phys. Soc. Japan 89 (2), 024802.
Khedmati, M., Seifi, F., Azizi, M., 2020. Time series forecasting of bitcoin price based on autoregressive integrated moving average and machine learning approaches. Int. J. Eng. 33 (7), 1293–1303.
Kończal, J., Wronka, M., 2024. Tail and memory behaviour of the cryptocurrency prices and stock market indices. Math. Appl. 52 (1).
Koutmos, D., 2023. Investor sentiment and bitcoin prices. Rev. Quant. Financ. Account. 60 (1), 1–29.
Kristoufek, L., Vosvrda, M., 2019. Cryptocurrencies market efficiency ranking: Not so straightforward. Phys. A 531, 120853.
Kumar, H., Patil, S.B., 2015. Estimation & forecasting of volatility using ARIMA, ARFIMA and neural network based techniques. In: 2015 IEEE International Advance Computing Conference. IACC, IEEE, pp. 992–997.
Le Tran, V., Leirvik, T., 2020. Efficiency in the markets of crypto-currencies. Financ. Res. Lett. 35, 101382.
Ledesma-Orozco, S., Ruiz-Pinales, J., García-Hernández, G., Cerda-Villafaña, G., Hernández-Fusilier, D., 2011. Hurst parameter estimation using artificial neural networks. J. Appl. Res. Technol. 9 (2), 227–241.
Li, Z., Han, J., Song, Y., 2020a. On the forecasting of high-frequency financial time series based on ARIMA model improved by deep learning. J. Forecast. 39 (7), 1081–1097.
Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., Talwalkar, A., 2020b. A system for massively parallel hyperparameter tuning. Proc. Mach. Learn. Syst. 2, 230–246.
Lian, J., 2024. Comparative analysis of LSTM, GRU and transformer deep learning models for cryptocurrency ZEC price prediction performance. In: 9th International Conference on Financial Innovation and Economic Development. ICFIED 2024, Atlantis Press, pp. 396–405.
Lian, Y.-M., Chen, J.-L., Cheng, H.-C., et al., 2022. Predicting bitcoin prices via machine learning and time series models. J. Appl. Financ. Bank. 12 (5), 25–43.
Liu, Y., Zhao, C., Huang, Y., 2022. A combined model for multivariate time series forecasting based on MLP-feedforward attention-LSTM. IEEE Access 10, 88644–88654.
Livieris, I.E., Stavroyiannis, S., Pintelas, E., Pintelas, P., 2020. A novel validation framework to enhance deep learning models in time-series forecasting. Neural Comput. Appl. 32 (23), 17149–17167.
Lommers, K., Harzli, O.E., Kim, J., 2021. Confronting machine learning with financial research. arXiv preprint arXiv:2103.00366.
López-Martín, C., Benito Muela, S., Arguedas, R., 2021. Efficiency in cryptocurrency markets: New evidence. Eurasian Econ. Rev. 11 (3), 403–431.
Luo, D., Mishra, T., Yarovaya, L., Zhang, Z., 2021. Investing during a fintech revolution: Ambiguity and return risk in cryptocurrencies. J. Int. Financ. Mark. Inst. Money 73, 101362.
Maciel, L., 2020. Technical analysis based on high and low stock prices forecasts: Evidence for Brazil using a fractionally cointegrated VAR model. Empir. Econ. 58 (4), 1513–1540.
Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2018. Statistical and machine learning forecasting methods: Concerns and ways forward. PLoS One 13 (3), e0194889.
McNally, S., Roche, J., Caton, S., 2018. Predicting the price of bitcoin using machine learning. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing. PDP, IEEE, pp. 339–343.
Mirzaei, M.M., Mizanian, K., Rezaeian, M., 2014. Modeling of self-similar network traffic using artificial neural networks. In: 2014 4th International Conference on Computer and Knowledge Engineering. ICCKE, IEEE, pp. 741–746.
Mishra, T., Park, D., Parhi, M., Uddin, G.S., Tian, S., 2023. A memory in the bond: Green bond and sectoral investment interdependence in a fractionally cointegrated VAR framework. Energy Econ. 121, 106652.
Mukherjee, S., Sadhukhan, B., Das, A.K., Chaudhuri, A., 2023. Hurst exponent estimation using neural network. Int. J. Comput. Sci. Eng. 26 (2), 157–170.
Mullainathan, S., Spiess, J., 2017. Machine learning: an applied econometric approach. J. Econ. Perspect. 31 (2), 87–106.
Noda, A., 2021. On the evolution of cryptocurrency market efficiency. Appl. Econ. Lett. 28 (6), 433–439.
Nosratabadi, S., Mosavi, A., Duan, P., Ghamisi, P., Filip, F., Band, S.S., Reuter, U., Gama, J., Gandomi, A.H., 2020. Data science in economics: comprehensive review of advanced machine learning and deep learning methods. Mathematics 8 (10), 1799.
Núñez, J.A., Contreras-Valdez, M.I., Franco-Ruiz, C.A., 2019. Statistical analysis of bitcoin during explosive behavior periods. PLoS One 14 (3), e0213919.
Ouandlous, A., Barkoulas, J.T., Pantos, T.D., 2022. Extremity in bitcoin market activity. J. Econ. Asymmetries 26, e00270.
Parmezan, A.R.S., Souza, V.M., Batista, G.E., 2019. Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model. Inform. Sci. 484, 302–337.
Parray, I.R., Khurana, S.S., Kumar, M., Altalbe, A.A., 2020. Time series data analysis of stock price movement using machine learning techniques. Soft Comput. 24, 16509–16517.
Penmetsa, S., Vemula, M., 2023. Cryptocurrency price prediction with LSTM and transformer models leveraging momentum and volatility technical indicators. In: 2023 IEEE 3rd International Conference on Data Science and Computer Application. ICDSCA, IEEE, pp. 411–416.
Peters, E.E., 1994. Fractal Market Analysis: Applying Chaos Theory to Investment and Economics, vol. 24, John Wiley & Sons.
Pirani, M., Thakkar, P., Jivrani, P., Bohara, M.H., Garg, D., 2022. A comparative analysis of ARIMA, GRU, LSTM and BiLSTM on financial time series forecasting. In: 2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics. ICDCECE, IEEE, pp. 1–6.
Qian, X.-Y., Gao, S., 2017. Financial series prediction: Comparison between precision of time series models and machine learning methods. pp. 1–9, arXiv preprint arXiv:1706.00948.
Rasmussen, C.E., 2003. Gaussian processes in machine learning. In: Summer School on Machine Learning. Springer, pp. 63–71.
Reddy, V.K.S., Sai, K., 2018. Stock market prediction using machine learning. Int. Res. J. Eng. Technol. (IRJET) 5 (10), 1033–1035.
Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65 (6), 386.
Rosenblatt, F., 1961. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books.
Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. De Rumelhart and J. Mcclelland vol. 1. 1986. Biometrika 71 (599–607), 6.
Rundo, F., Trenta, F., Di Stallo, A.L., Battiato, S., 2019. Machine learning for quantitative finance applications: A survey. Appl. Sci. 9 (24), 5574.
Ryll, L., Seidens, S., 2019. Evaluating the performance of machine learning algorithms in financial market forecasting: A comprehensive survey. arXiv preprint arXiv:1906.07786.
dos Santos Maciel, L., Ballini, R., 2019. On the predictability of high and low prices: The case of bitcoin. Braz. Rev. Financ. 17 (3), 66–84.

Sapankevych, N.I., Sankar, R., 2009. Time series prediction using support vector machines: a survey. IEEE Comput. Intell. Mag. 4 (2), 24–38.
Sapkota, N., Grobys, K., 2021. Asset market equilibria in cryptocurrency markets: Evidence from a study of privacy and non-privacy coins. J. Int. Financ. Mark. Inst. Money 74, 101402.
Sermpinis, G., Stasinakis, C., Dunis, C., 2014. Stochastic and genetic neural network combinations in trading and hybrid time-varying leverage effects. J. Int. Financ. Mark. Inst. Money 30, 21–54.
Shanaev, S., Ghimire, B., 2021. A fitting return to fitting returns: cryptocurrency distributions revisited.
Shimotsu, K., 2010. Exact local whittle estimation of fractional integration with unknown mean and time trend. Econometric Theory 26 (2), 501–540.
Shimotsu, K., 2012. Exact local whittle estimation of fractionally cointegrated systems. J. Econometrics 169 (2), 266–278.
Shimotsu, K., Phillips, P.C., 2005. Exact local whittle estimation of fractional integration.
Siami-Namini, S., Tavakoli, N., Namin, A.S., 2018. A comparison of ARIMA and LSTM in forecasting time series. In: 2018 17th IEEE International Conference on Machine Learning and Applications. ICMLA, IEEE, pp. 1394–1401.
Singh, S., Bhat, M., 2024. Transformer-based approach for ethereum price prediction using crosscurrency correlation and sentiment analysis. arXiv preprint arXiv:2401.08077.
Smilkov, D., Carter, S., 2023. Https://playground.tensorflow.org/. URL https://playground.tensorflow.org/.
Soni, P., Tewari, Y., Krishnan, D., 2022. Machine learning approaches in stock price prediction: A systematic review. In: Journal of Physics: Conference Series. 2161, (1), IOP Publishing, 012065.
Sørensen, N.H., 2023. Comparing GARCH and NN for forecasting TTF volatility (Ph.D. thesis). Master Thesis. Aalborg University Business School.
Sridhar, S., Sanagavarapu, S., 2021. Multi-head self-attention transformer for dogecoin price prediction. In: 2021 14th International Conference on Human System Interaction. HSI, IEEE, pp. 1–6.
Tang, Y., Song, Z., Zhu, Y., Yuan, H., Hou, M., Ji, J., Tang, C., Li, J., 2022. A survey on machine learning models for financial time series forecasting. Neurocomputing 512, 363–380.
Tanwar, A., Kumar, V., 2022. Prediction of cryptocurrency prices using transformers and long short term neural networks. In: 2022 International Conference on Intelligent Controller and Computing for Smart Power. ICICCSP, IEEE, pp. 1–4.
Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B Stat. Methodol. 58 (1), 267–288.
Tjøstheim, D., 2025. Selected topics in time series forecasting: Statistical models vs. machine learning. Entropy 27 (3), 279.
Ubal, C., Di-Giorgi, G., Contreras-Reyes, J.E., Salas, R., 2023. Predicting the long-term dependencies in time series using recurrent artificial neural networks. Mach. Learn. Knowl. Extr. 5 (4), 1340–1358.
Urquhart, A., 2016. The inefficiency of bitcoin. Econom. Lett. 148, 80–82.
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Adv. Neural Inf. Process. Syst. 30.
Wei, W.C., 2018. Liquidity and market efficiency in cryptocurrencies. Econom. Lett. 168, 21–24.
Wu, L., Chen, S., 2020. Long memory and efficiency of bitcoin under heavy tails. Appl. Econ. 52 (48), 5298–5309.
Yan, H., Ouyang, H., 2018. Financial time series prediction based on deep learning. Wirel. Pers. Commun. 102, 683–700.
Yarovaya, L., Matkovskyy, R., Jalan, A., 2021. The effects of a "black swan" event (COVID-19) on herding behavior in cryptocurrency markets. J. Int. Financ. Mark. Institutions Money 75, 101321.
Yaya, O.S., Vo, X.V., Ogbonna, A.E., Adewuyi, A.O., 2022. Modelling cryptocurrency high–low prices using fractional cointegrating VAR. Int. J. Financ. Econ. 27 (1), 489–505.
Zhang, C., Sjarif, N.N.A., Ibrahim, R., 2024. Deep learning models for price forecasting of financial time series: A review of recent advancements: 2020–2022. Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 14 (1), e1519.
Zhao, J., Huang, F., Lv, J., Duan, Y., Qin, Z., Li, G., Tian, G., 2020. Do RNN and LSTM have long memory? In: International Conference on Machine Learning. PMLR, pp. 11365–11375.
Zheng, W., Zhao, P., Huang, K., Chen, G., 2021. Understanding the property of long term memory for the LSTM with attention mechanism. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. pp. 2708–2717.