Stock Price Time Series Forecasting Using Dynamic Graph Neural Networks and Attention Mechanism in Recurrent Neural Networks

Marco Gregnanin[1,2][0009−0004−3205−6997], Johannes De Smedt[2][0000−0003−0389−0275], Giorgio Gnecco[1][0000−0002−5427−4328], and Maurizio Parton[3][0000−0003−4905−3544]

[1] IMT School for Advanced Studies Lucca, Italy
[2] KU Leuven, Belgium
[3] University of Chieti-Pescara, Italy

Abstract. Forecasting stock prices is a challenging problem in financial markets. Traditional statistical time series models often struggle to achieve satisfactory results, while deep learning models have shown promise to provide even better outcomes. These models typically focus on capturing and exploiting temporal dependencies within the time series data.

In this study, we propose a novel model that incorporates both temporal and geometric patterns present in stock price time series data. Firstly, we employ the visibility graph theory to derive a graph representation of the stock price time series, enabling us to capture the inherent non-Euclidean relationships within the data, which is important because it could lead to a better representation of the observations. Subsequently, we utilize Graph Neural Networks (GNNs) to analyze and model these relationships effectively. Finally, we incorporate a Long Short-Term Memory (LSTM) network enhanced with an attention mechanism to capture the temporal dependencies.

Through numerical evaluation, our proposed model demonstrates superior performance compared to state-of-the-art models in the prediction of stock prices. The integration of a graph-based representation, GNNs, LSTM, and an attention mechanism enables our model to effectively capture both the temporal and geometric patterns of the stock price time series data.

This research contributes to the field of stock price prediction by introducing a comprehensive model that is able to combine the strengths of GNNs and the LSTM network to exploit non-linear dependencies and patterns inside a time series to address the limitations current Recurrent Neural Networks (RNNs).

Keywords: Dynamic Graph Neural Networks · Time Series prediction · Visibility Graph.

1 Introduction

Predicting future values in a time series poses a significant challenge, particularly in the context of financial time series characterized by their stochastic behavior. Classical statistical models, such as Vector Auto Regressive (VAR) or Auto Regressive Integrated Moving Average (ARIMA) [1], encounter difficulties in effectively addressing the regression task due to the presence of noise and non-linearity within stock prices. Similarly, financial mathematical models like the Brownian motion [2] or more generalized diffusion processes [3] struggle to accurately replicate the market behaviour of financial time series. In contrast, deep learning models demonstrate promising results by leveraging their ability to capture and exploit non-linearities within the data. Although classical Recurrent Neural Networks (RNNs) [4] excel at identifying temporal patterns within time series, relying solely on temporal dependencies may prove insufficient for achieving high predictive accuracy when dealing with financial time series.
We contend that geometric patterns play a crucial role in forecasting problems related to this type of time series. Specifically, transforming the time series from a Euclidean space to a non-Euclidean space using a graph-based approach can aid in capturing and analyzing the inherent complexity of financial time series. By examining the degree distribution of the graph's nodes, we can determine whether the series exhibits periodic, random, or fractal characteristics [5]. This is particularly relevant in financial time series since fractal series, characterized by self-similarity and repetition at different scales, are common in stock prices. Incorporating fractal concepts into financial modeling can result in a more precise representation of market dynamics [6]. Graph Neural Networks (GNNs) possess the capacity to capture and model geometric patterns that are inherent in graph data. Like fractals, GNNs can capture local structure information within a graph by iteratively aggregating information from neighboring nodes. This recursive process enables GNNs to learn and generalize patterns at various scales, thereby effectively capturing the inherent geometric properties of the graph as fractals do with self-similarity.
To address this challenge, we propose a novel forecasting model that effectively harnesses both temporal and geometric patterns within a financial time series. Our contributions are:

(1) the introduction of an architecture that combines GNNs [7] with Long Short-Term Memory (LSTM) models enhanced with an attention mechanism;
(2) showing the relationship between the Mean Absolute Scale Error (MASE) [8], which serves as an evaluation metric, and the Efficient Market Hypothesis (EMH) [9], which postulates the efficiency of financial markets in reflecting all available information in the stock prices;
(3) an evaluation of our approach on the Standard and Poor 500 (S&P500) stock exchange dataset, showcasing improved prediction results compared to benchmark models.

The remainder of this paper is organized as follows: Section 2 reviews statistical time series models and prevalent neural network models employed in the

stock price prediction literature; Section 3 provides definitions of both static and dynamic graphs, explores the visibility graph methodology for deriving graphs from univariate time series, and shows the concepts of the LSTM network and the attention mechanism; Section 4 details the architecture of our proposed model; Section 5 compares the performance of our model against the benchmark models considered; and finally, Section 6 concludes the paper.

2 Related Work

The forecasting problem for time series can be approached using either statistical models or deep learning models. Statistical models, such as ARIMA and VAR, perform well when the time series is stationary [1, 10, 11], indicating the absence of hidden patterns that could alter statistical properties. ARIMA captures linear dependencies within a time series, while VAR extends this concept to capture linear interdependencies among multiple time series. In previous studies, the ARIMA model was employed to predict stock prices on the New York Stock Exchange (NYSE) and the Nigeria Stock Exchange (NSE) [12], as well as the National Fifty (NIFTY50) index using a non-seasonal ARIMA model [13]. The VAR model was proposed for one-step ahead prediction of six different stock exchanges [14]. In the context of volatility prediction, in [15] a comparison was made between ARIMA and the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model for various stock indices. However, these models face challenges in capturing non-linear patterns in financial time series.

On the other hand, deep learning models have shown promise in capturing non-linear patterns in time series [16, 17]. In the prediction of stock prices, comparisons were made among LSTM, RNNs, and Convolutional Neural Networks (CNNs) for companies listed on the Stock Exchange of India [18]. LSTM models were used to predict one-step ahead values of the NIFTY50 index based on daily 'Open, Close, High, Low' prices [19]. To enhance prediction accuracy, the State Frequency Memory (SFM) model incorporated the Discrete Fourier Transform (DFT) within LSTM memory cells to capture patterns at different frequencies [20]. An ensemble of methods including CNNs, Bidirectional LSTM (BiLSTM), and an attention mechanism was proposed for predicting the next day's price of the Shanghai Composite Index (SSE) [21, 22]. Combining different layers of LSTM with an attention mechanism was suggested to forecast future values of stock exchange prices for six markets [23]. GNN models were employed to analyze news and macroeconomic factors associated with companies to improve prediction accuracy [24, 25]. Furthermore, an ensemble model combining BiLSTM and Graph Convolutional Networks (GCNs) was proposed for one-step ahead prediction of oil prices [26].

Finally, geometric deep learning models are employed to capture geometric patterns when data can be represented in a non-Euclidean space. These models are typically focused on various domains, including social network analysis, recommender systems, traffic forecasting, chemistry and biology data, and graph classification [27]. Such models can be categorized as either static or dynamic, de-

pending on the behavior of the underlying graph data. In the former case, GCNs have been introduced for classification problems on citation networks and knowledge graphs [28]. The GraphSAGE (SAmple and aggreGatE) model [29] and the Graph Attention Network (GAT) [30] have been proposed for the classification of citation networks and protein-protein interaction datasets. On the other hand, Dynamic Graph Neural Networks (DGNNs) have been developed to capture the temporal dynamics and evolving structures in dynamic graph data, such as social networks, traffic, energy, and electrocardiogram datasets [31]. The Temporal Graph Convolutional Networks (T-GCN) [32] and the Graph WaveNet (GWN) model [33] have been specifically designed for traffic forecasting. Additionally, the Graph Recurrent Neural Networks (GRNNs) [34] have demonstrated their effectiveness in earthquake epicenter estimation, traffic forecasting, and epidemic tracking analyses.

However, none of these models have attempted to exploit both the temporal and geometric patterns present in financial time series data. Additionally, existing works utilizing GNNs have mainly considered static graphs [26], and have derived the graph from corporate information [24, 25]. Our proposed model aims to capture non-linear patterns by dynamically adapting the graph structure of the data, thereby generating accurate predictions of stock prices.

3 Preliminaries

Network theory. Let $G = (V, E)$ be a graph, where $V = \{v_1, \ldots, v_n\}$ is the set of nodes, and $E \subseteq V \times V$ is the set of edges. The adjacency matrix A of the associated graph G is an $n \times n$ matrix, whose entries are 1 or 0 indicating whether nodes u and v are connected by an edge, i.e., $A(u, v) = 1$ if and only if $(u, v) \in E$.
In this paper, we define a dynamic graph using a snapshot-based representation [35]. Thus, a dynamic graph is a sequence of static graph snapshots in discrete time with a set $\mathcal{T} = \{t_1, t_2, \ldots, t_T\}$ of $T = |\mathcal{T}|$ time steps, i.e., $\mathcal{G} = \{G_t\}_{t=1}^{T}$, where $G_t = (V_t, E_t)$ with $V_t \subseteq V$ and $E_t \subseteq E$. The sets of nodes and edges can vary in each graph snapshot.
To derive a graph from a time series, we employ the visibility graph algorithm [5], which connects data points if there is a clear line of sight between them. The visibility graph method is based on the idea of representing the values of a time series as vertical bins. It involves establishing connections between each bar and the bars that can be seen clearly from the top, without any other bars obstructing the view. In this approach, each data point becomes a node, and the edges represent visibility relationships. Given a time series $S = \{s_1, s_2, \ldots, s_T\}$ and the set of associated discrete time stamps $\mathcal{T} = \{t_1, t_2, \ldots, t_T\}$, we define the following visibility criterion: any two data points (s_i, t_i) and (s_j, t_j) are connected in the graph if any other data point (s_k, t_k) satisfies the following condition:

$$s_k < s_j + (s_j - s_i)\frac{t_j - t_k}{t_j - t_i}\,. \tag{1}$$

The visibility graph algorithm captures the non-linear dependencies and patterns in the time series data. Moreover, the resulting graph from the time series is connected, and invariant under affine transformations of the time series data. The visibility graph method allows for the derivation of either undirected or directed graphs. In the case of the former, when applying the visibility condition, all possible combinations of values $(s_i, t_i), i = 1, \ldots, T$ are considered. For the directed graph, the visibility condition is applied following the "from left to right" rule, meaning that the value (s_i, t_i) is compared with all other values associated with discrete time stamps greater than t_i, i.e., $\forall (s_j, t_j), j = i + 1, \ldots, T$.
For our analysis, we have chosen the case of the undirected graph. Figure 1 shows an example of how the visibility graph algorithm works on a univariate time series. The first plot depicts the trajectories of the closing prices. The middle plot illustrates the visibility condition and the creation of links. The final plot represents the corresponding undirected graph, which is computed using the Python package "Time series to visibility graphs" (ts2vg) [36]. Furthermore, this algorithm preserves several properties of the series in the graph representation. For instance, a periodic time series is transformed into a regular graph, while a random time series becomes a random graph.

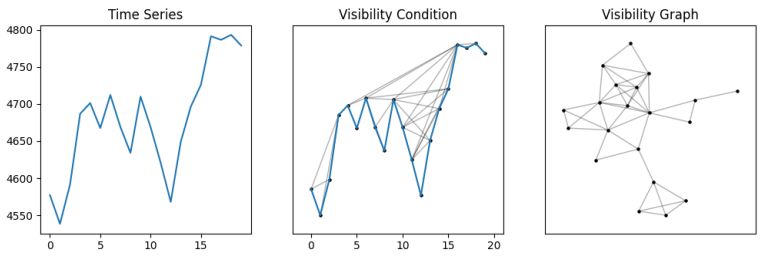

Fig. 1. Visibility graph method on the Standard and Poor's 500 closing price from 02/12/2021 to 31/12/2021.

Recurrent Neural Networks. Hochreiter and Schmidhuber introduced the Long Short-Term Memory (LSTM) architecture [37], a specialized type of RNN that can capture long-term dependencies in a time series and alleviate the vanishing gradient problem. Formally, the LSTM model is based on the input gate, forget gate, output gate, cell state, and hidden state, defined as follows:

$$
\begin{aligned}
\mathbf{i}_t &= \mathit{Sigmoid}(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i)\,, \\
\mathbf{f}_t &= \mathit{Sigmoid}(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f)\,, \\
\mathbf{o}_t &= \mathit{Sigmoid}(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o)\,, \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathit{tanh}(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c)\,, \\
\mathbf{h}_t &= \mathbf{o}_t \odot \mathit{tanh}(\mathbf{c}_t)\,,
\end{aligned}
\tag{2}
$$

where $\mathbf{x}_t$ is the vector of input features at time t, $\odot$ denotes the Hadamard (entrywise) product, W_i, W_f, W_o, W_c and U_i, U_f, U_o, U_c are weight matrices, $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o, \mathbf{b}_c$ are bias vectors, and the three gates control the flow of information in the LSTM architecture.

Attention Mechanism. The attention mechanism dynamically focuses on relevant parts of input sequences [38, 39]. The attention mechanism operates by assigning distinct weights to various components of the input, enabling the model to concentrate on the most pertinent information for extracting essential patterns. This mechanism facilitates the extraction of crucial features by selectively focusing on relevant input elements, enhancing the model's performance in capturing important patterns and dependencies. Formally, the attention score can be defined as follows:

$$\begin{aligned} e_{t,i} &= a(\mathbf{o}_{t-1}, \mathbf{h}_i)\,, \\ \alpha_{t,i} &= \left(\mathit{Softmax}(\mathbf{e}_{t,\cdot})\right)_i\,, \\ \mathbf{d}_t &= \sum_{i=1}^{T} \alpha_{t,i} h_i\,, \end{aligned} \tag{3}$$

where the vector $\mathbf{h}_i$ is the hidden state, the vector $\mathbf{o}_{t-1}$ is the previous output, $a(\cdot)$ is an alignment function (typically, the inner product), and $\mathbf{d}_t$ is the final attention score. It is important to note that the *Softmax* function operates on a vector input, denoted as $\mathbf{e}_{t,\cdot}$ in our context, where the index t remains fixed while the other index varies freely.

Graph Neural Networks. GNNs are deep learning models that learn node embeddings for graphs. GNNs are based on a message-passing framework, which allow information to be propagated among nodes in a graph [40, 41]. Following the notation in [42], we define the l-th layer in the GNN as:

$$\begin{aligned} m_{u\to v}^{(l)} &= \mathrm{MSG}^{(l)}(h_u^{(l-1)}, h_v^{(l-1)})\,, \\ h_v^{(l)} &= \mathrm{AGG}^{(l)}(m_{u\to v}^{(l)} | u \in N(u), {h_v}^{(l-1)})\,, \end{aligned} \tag{4}$$

where $h_v^{(l)}$ corresponds to the node v embedding and is initialized with the feature value of node v, i.e., $h_v^{(0)} = x_v$, $\mathrm{MSG}(\cdot)$ is the message function, $m^{(l)}$ is the message embedding at the l-th layer, $N(u)$ represents the set of neighboring nodes of node u, and $\mathrm{AGG}(\cdot)$ is the aggregation function, will be explained later. The objective of the message function is to facilitate communication and information exchange among nodes and their neighbors. One common approach is to employ a linear function to compute the message. On the other hand, the aggregation function combines the received messages from neighboring nodes into a unified representation for the receiving nodes. It is crucial for the aggregation function to be permutation invariant and produce equivalent node representations. Common examples of aggregation functions include the mean, sum, and maximum functions. The notation "l-layer" denotes the information originating

from nodes that are at a distance of l-hops away. This notation captures the concept of information propagation across multiple layers in a GNN, enabling the incorporation of increasingly larger neighborhoods into the node representations. By changing the definition of the message-passing framework, one can modify the type of GNN model employed.

4 Proposed Framework

Let $\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_T\}$ represent a multivariate time series, where each element $\mathbf{s}_t = [s_{1,t}, s_{2,t}, \ldots, s_{F,t}]$ is a vector of dimension F, capturing the observations at time step t. Here, F denotes the number of features that describe the multivariate time series. The objective of this research is to determine a mapping function, denoted as f, which can predict the future q values $s_{t+1:t+q}$ of a univariate target time series selected from $\mathbf{S}$, denoted with S. This prediction is based on a graph associated with the target time series and a feature matrix, both computed at time t. The prediction function can be expressed as follows:

$$s_{t+1:t+q} = f(G_t, X_t)\,, \tag{5}$$

where G_t denote the graph representation of the univariate target time series S, which incorporates the past m observations up to time t, and the current observation. Specifically, we represent this set of observations as $S_t = \{s_{t-m}, \ldots, s_t\}$, where S_t consists of $m+1$ elements. Similarly, the feature matrix X_t is constructed by arranging the last $m+1$ observations coming from each univariate times series in $\mathbf{S}$ in a tabular format, where each row corresponds to a specific time step and each column represents a distinct feature. Thus, the dimension of the feature matrix X_t is $(m+1) \times F$, where $m+1$ is the number of time steps and F is the number of features. For notational convenience, we introduce $\tilde{T}$ to denote $m+1$. Thus, the feature matrix X_t can be expressed as $X_t \in \mathbb{R}^{\tilde{T} \times F}$.
For solving this regression problem, we propose a model that combines GNNs with an LSTM layer enhanced by an attention mechanism. As a baseline model, we consider the AT-LSTM model proposed in [23]. The AT-LSTM model incorporates LSTM layers with an attention mechanism, consisting of two LSTM layers, followed by an attention mechanism, and another LSTM layer before two fully connected layers. In addition, we include in the comparison a vanilla GCN with two layers, followed by a single layer of LSTM, a BiLSTMs model, and the ARIMA model.

4.1 TA-DGNN: Temporal Attention - Dynamic Graph Neural Network

The proposed model combines the attention mechanism and the LSTM model with DGNNs. The architecture of the proposed model is illustrated in Figure 2. We begin by considering the feature matrix $X_t \in \mathbb{R}^{\tilde{T} \times F}$ and the set of observations $S_t = \{s_{t-m}, \ldots, s_t\}$ from the target time series S. Subsequently, we

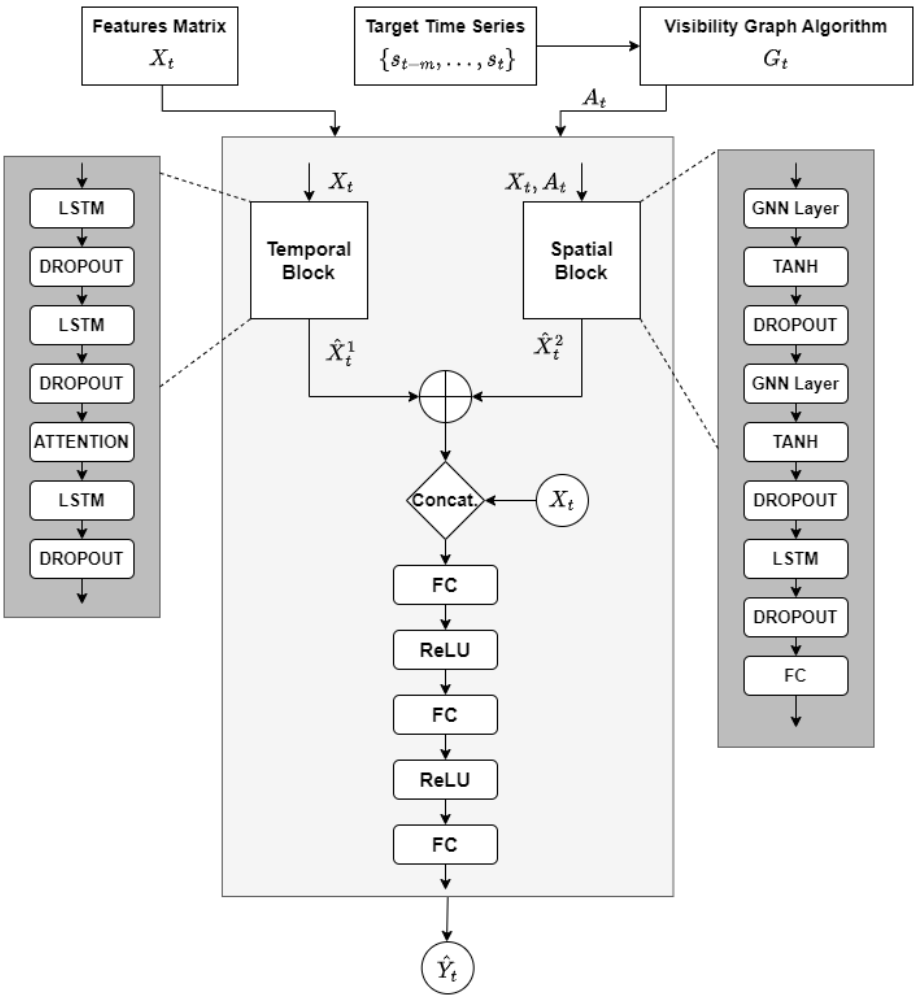

Fig. 2. Architecture of the proposed TA-DGNN model.

construct the undirected graph G_t using the visibility graph algorithm from the set S_t. This approach ensures that the spatial patterns considered in the graph construction are solely derived from the target series that we aim to predict, i.e. S. This graph has the associated symmetric adjacency matrix $A_t \in \mathbb{R}^{\tilde{T}\times\tilde{T}}$, where each node corresponds to a time observation. As illustrated in Figure 2, the architecture of our proposed model can be divided into three sections: the temporal block, the spatial block, and several fully connected layers.

Temporal Block. The temporal block receives the feature matrix X_t as input, which undergoes two layers of LSTM to capture temporal and long-term patterns. The output after these two layers has dimension $\tilde{T} \times F$. A dropout layer is placed between the LSTM layers to prevent overfitting, followed by another dropout layer before passing the data to the attention layer. The attention layer assigns different weights to input features based on their importance, enabling the identification of informative input components. Consequently, the primary objective of the attention mechanism is to capture and model the relationships among these features. The output of the attention layer has dimension $\tilde{T} \times F$. Finally, another LSTM layer, followed by a dropout layer, processes the patterns identified by the attention layer. The resulting output of the temporal block is denoted as $\hat{X}_t^1 \in \mathbb{R}^{\tilde{T}\times F}$.

Spatial Block. The spatial block takes the feature matrix X_t and the adjacency matrix A_t as input. Initially, a GNN layer is applied, followed by the hyperbolic tangent ($tanh$) activation function and a dropout layer. This GNN layer captures information from neighboring nodes that are 1-hop away. The output dimension of this initial three-block is $\tilde{T} \times K_1$, where K_1 is the dimension of the weight matrix of the GNN layer (i.e., the number of neurons). The same sequence of layers is then repeated. The second GNN layer captures information from nodes that are 2-hops away from the original nodes. The output dimension of the second three-block is $\tilde{T} \times K_2$, where K_2 is the dimension of the weight matrix of the second GNN layer. Before applying a dropout layer, the data are passed through an LSTM layer to process the spatial patterns identified by the two GNN layers. The output dimension of the LSTM layer is $\tilde{T} \times K_3$, where K_3 represents the number of neurons in the LSTM layer. Finally, a fully connected layer is applied, resulting in the final output of the spatial block denoted as $\hat{X}_t^2 \in \mathbb{R}^{\tilde{T}\times F}$.

Our model allows for the utilization of various types of GNN layers. In this study, we consider three types: Graph Convolutional Network (GCN) [28], Chebyshev Spectral Graph Convolution [43], and GraphSAGE Convolutional Neural Network [29]. Notation-wise, we provide the general definitions without the time subscript and combine the activation function for simplicity.

For the GCN, each hidden layer is computed as $H = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}XW\right)$, where $\tilde{A} = A + I \in \mathbb{R}^{\tilde{T}\times\tilde{T}}$ is the adjacency matrix with self-loops, $\tilde{D} = \sum_j \tilde{A_{ij}} \in \mathbb{R}^{\tilde{T}\times\tilde{T}}$ is the degree matrix, $X \in \mathbb{R}^{\tilde{T}\times F}$ is the input matrix, $W \in \mathbb{R}^{F\times K}$ is the

weight matrix, $\sigma(\cdot)$ is the activation function, $\tilde{T}$ is the number of nodes in the graph, F is the number of features, and K is the output dimension of the layer.

For the Chebyshev Spectral Graph Convolution, each hidden layer is defined as $H = \sigma\left(\sum_{r=1}^{R} Z^{(r)} W^{(r)}\right)$, where R represents the Chebyshev filter size, $W^{(r)}$ is the weight matrix, and $Z^{(r)}$ is recursively computed as follows: $Z^{(1)} = X, Z^{(2)} = \hat{L}X, Z^{(k)} = 2\hat{L}Z^{(r-1)} - Z^{(r-2)}$. Here, $\hat{L}$ denotes the adjusted Laplacian matrix defined as $\hat{L} = \frac{2L}{\lambda_{max}} - I$, where λ_{max} is the maximum eigenvalue of the Laplacian matrix L.

For the GraphSAGE Convolutional Neural Network, the activation of the i-th neuron in each hidden layer is defined as $H_i = \sigma\left(W_1 \mathbf{x}_i + W_2 \text{mean}_{j \in N(i)} \mathbf{x}_j\right)$, where W_1 and W_2 are the weight matrices, $N(i)$ represents the set of neighborhood nodes of node i.

Fully Connected Layers. The outputs of the temporal and spatial blocks, namely $\hat{X}_t^1$ and $\hat{X}_t^2$, are processed in the final part of the model. First, the two outputs are summed and concatenated with the feature matrix X_t, resulting in an output dimension of $2\tilde{T} \times F$. Finally, three fully connected layers with 128, 64, and q cells, respectively, are applied. The Rectified Linear Unit (ReLU) activation function $g(x) = \max(x, 0)$ [44] is used in the fully connected layers to capture the final non-linear patterns.

The model's final output, denoted as $\hat{Y}_t$, represents the predicted value of the target variable for the future q steps.

5 Numerical comparisons

In this section, we provide a detailed account of the model comparison methodology employed. Specifically, we begin by discussing the nature of the dataset utilized for the analysis. Subsequently, we introduce the evaluation metrics employed for the purpose of comparison. Prior to delving into the presentation of the model results, we establish the hyperparameters for both the proposed model and the baseline models.

5.1 Dataset

For our analysis, we focus on the Standard and Poor's 500 (S&P 500) stock exchange. We have chosen to focus on the S&P 500 index due to its high liquidity and efficiency as a stock market [45, 46]. This index encompasses 500 large publicly traded companies from various industries and sectors in the United States market [47]. Additionally, the S&P 500 holds significant influence in economic studies and serves as a reflection of the performance of the United States economy [48]. We collect daily price observations of Close, Open, High, and Low from Yahoo Finance [49] spanning from Monday 4$^{\text{th}}$ January, 2010 to Sunday 4$^{\text{th}}$ June, 2023, resulting in 3,377 observations.

Considering Equation (5), we can reframe the regression problem by setting q equal to 1 and m equal to 20. We chose 20 days as the length of the window since the average number of trading days in a month is approximately 21 in the United States financial market. Therefore, our prediction function can be expressed as:

$$s_{t+1} = f(G_t, X_t)\,,$$

where $X_t \in \mathbb{R}^{20\times 4}$ is the feature matrix with four features, i.e, Close, Open, High, and Low prices, and G_t is the graph derived from the closing price considering the previous 20 observations, i.e., $S_t = \{s_{t-20}, \ldots, s_t\}$. Our objective is to predict the closing price for the next day.

To effectively capture the non-linear patterns within the time series, the neural network models require a normalized dataset. Thus, we normalize our dataset using the following method:

$$\tilde{x}_t^i = \frac{x_t^i - \mu_t^i}{\sigma_t^i}\,, \tag{6}$$

where $i \in \{\textit{Close, Open, High, Low}\}$, x_t^i is the value of the i-th feature at time t, μ_t^i and σ_t^i represent the mean and standard deviation of the corresponding feature computed using a sliding window of 20 days.

Finally, we split our dataset into training, validation, and test sets according to the following distribution: 70% for training, 10% for validation, and 20% for testing. We include a validation set to incorporate an early stopping condition during the model training phase, set to 20 epochs, to prevent overfitting.

5.2 Evaluation Metrics

In order to compare the results among different models, we have selected several evaluation metrics, including Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Mean Absolute Scaled Error (MASE) [8]. These metrics are defined as follows:

$$\begin{aligned}
\text{MSE} &= \frac{1}{l}\sum_{i=1}^{l}(y_i - \hat{y}_i)^2\,, & \text{RMSE} &= \sqrt{\frac{1}{l}\sum_{i=1}^{l}(y_i - \hat{y}_i)^2}\,, \\
\text{MAE} &= \frac{1}{l}\sum_{i=1}^{l}|y_i - \hat{y}_i|\,, & \text{MAPE} &= \frac{1}{l}\sum_{i=1}^{l}\left|\frac{y_i - \hat{y}_i}{y_i}\right|\,, \\
\text{MASE} &= \frac{\text{MAE}}{\frac{1}{l-1}\sum_{i=2}^{l}|y_i - y_{i-1}|}\,,
\end{aligned} \tag{7}$$

where l denotes the size of one among the training, validation, and test sets, y_i is the true value for the i-th example in the training, validation, or test set, and $\hat{y}_i$ is the corresponding output of the model.

The last two metrics, MAPE and MASE, are scale-free error metrics. The MASE

metric compares the actual forecast against a naive forecast where the future one-step ahead value is equal to the previous value. For all the evaluation metrics, smaller values indicate better performance of the models in predicting future values. In the case of the MASE metric, a value lower than 1 suggests that the model's forecast outperforms the naive forecast, while a value greater than 1 indicates the opposite.

In our opinion the MASE metric is particularly useful for verifying the validity of the Efficient Market Hypothesis (EMH) [9, 50]. Indeed, according to the EMH, it is not possible to consistently predict the future price of a stock, and the best prediction for the next day's price is simply the current price (i.e., the naive forecast). Therefore, if the EMH holds true, the MASE value of the model should be close to or greater than 1.

5.3 Configurations of the models

In order to train the models using the training set, it is necessary to define the loss function and set the values of the hyperparameters. We have chosen the Mean Squared Error (MSE) as the loss function and incorporated the l_2 regularization term to prevent overfitting. Additionally, we need to determine the batch size, number of epochs, optimization method, and learning rate. The batch size refers to the number of samples given to the model before updating its parameters, while the number of epochs represents the number of complete passes through the dataset. The optimization method and learning rate govern the learning process and its speed. For our study, we have set the batch size to 25, the number of epochs to 500, utilized the Adam optimizer [51], and assigned a learning rate of 0.0001. Moreover, the dropout rate is set equal to 0.1. These hyperparameters have been consistently applied across all the models considered in our study.
As a reminder, our baseline models for comparison encompass the vanilla GCN augmented with an LSTM layer, the BiLSTM model, the AT-LSTM model, and the ARIMA model. Specifically, for the former model, we utilize two GCN layers with 16 and 10 neurons, respectively, followed by an LSTM layer with 100 neurons, which are respectively used also for K_1, K_2, and K_3 parameters of the spatial block. The BiLSTM model consists of three layers, each comprising 50 neurons. The AT-LSTM model mirrors the structure of the temporal block shown in Figure 2, without the inclusion of dropout layers. Lastly, in the ARIMA model, we set the autoregressive component to 3, the differencing component to 0, and the moving average component to 2, denoted as ARIMA$(3, 0, 2)$. The optimal combination of parameters, i.e., the combination $(3, 0, 2)$, for the ARIMA model have been determined by utilizing the Akaike Information Criterion (AIC) [52] across various combinations of parameters.

5.4 Numerical results

Table 1 presents the results of our analysis, specifically showcasing the evaluation metrics computed on the test set. For all the metrics considered, a smaller value indicates better model performance.

Table 1. Model performance in terms of MSE, RMSE, MAE, MAPE and MASE for the test set. The best value for each metric is in bold.

Models	MSE	RMSE	MAE	MAPE	MASE
ARIMA$(3, 0, 2)$	2269.7678	47.64208	36.0789	0.8886 %	1.0137
GNN+LSTM	4742.4291	68.8653	55.5719	1.3396 %	1.5624
BiLSTM	9361.4256	96.7545	78.4831	1.9301 %	2.2066
AT-LSTM	1254.2723	35.4157	22.3470	0.5547 %	0.6276
TA-DGNN (GCN)	**123.4954**	**11.1128**	**7.6060**	**0.1881** %	**0.2136**
TA-DGNN (ChebConv)	216.5798	14.7166	9.5463	0.2371 %	0.2681
TA-DGNN (SAGE)	127.2517	11.2806	7.7775	0.1925 %	0.2184

Our proposed model, TA-DGNN, outperforms the baseline models across all metrics. The best performance is observed when utilizing the GCN layers within the TA-DGNN. Furthermore, it is worth noting the MASE metric. With the exception of the AT-LSTM model, all baseline models exhibit a value greater than or equal to 1. This implies that these models are unable to consistently predict future market prices. Conversely, for the remaining models, the MASE metric is less than 1, indicating that the future stock values can be predicted consistently.
In summary, our proposed TA-DGNN model, particularly with the inclusion of GCN layers, demonstrates superior performance compared to the baseline models across all metrics. The MASE metric further suggests that the EMH is not supported when our proposed model is utilized for predicting future stock values.

To highlight the differences in prediction results among the models, Figure 3 presents a plot depicting the model predictions against the true closing prices. A closer proximity of the line to the true values indicates a better model performance. For clarity, only a limited subset of the closing price values from the test set is displayed to illustrate the differences in values.

The final analysis we conducted involves our proposed model and the AT-LSTM baseline model, in the context of an out-of-sample test. We consider only the AT-LSTM model as a baseline because in the previous analysis it obtained the best results with respect to the BiLSTM, ARIMA, and GCN baseline models. Specifically, we aimed to evaluate the performance of our model during a bearish market situation, characterized by economic restrictions. This analysis was conducted to assess the robustness of our proposed model. To this end, we collected daily observations of the S&P500's Close, Open, High, and Low from Yahoo Finance [49], spanning from Monday 1st October, 2007 to Tuesday 1st

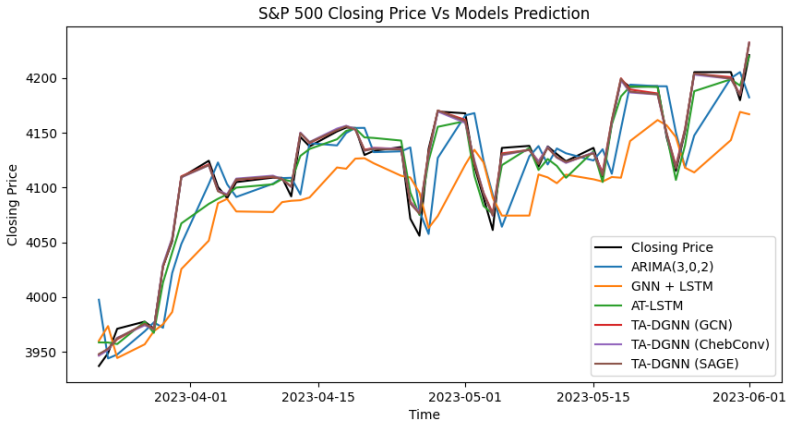

Fig. 3. Comparisons among the models' predictions and the S&P 500 closing price from 22/03/2023 to 01/06/2023.

December, 2009, resulting in 547 observations. This period corresponds to the financial crisis in the United States.
We preprocessed the data as for the previous analysis, and we kept the same forecasting horizon, i.e., one day. The results of the out-of-sample test are presented in Table 2, whereas Figure 4 illustrates the comparison between the true closing prices and the predicted values for a subset of the test set.

Table 2. Model performance in terms of MSE, RMSE, MAE, MAPE and MASE for the out-of-sample analysis. The best value for each metric is in bold.

Models	MSE	RMSE	MAE	MAPE	MASE
AT-LSTM	28.3803	5.3273	4.2145	0.4168 %	0.4682
TA-DGNN (GCN)	**5.5232**	**2.3501**	1.7421	0.1715 %	0.1935
TA-DGNN (ChebConv)	7.8564	2.8029	1.9653	0.1949 %	0.2183
TA-DGNN (SAGE)	5.7383	2.3955	**1.7336**	**0.1710** %	**0.1926**

As observed in Table 2, our proposed model continues to outperform the AT-LSTM baseline model. However, there are differences in performance across distinct evaluation metrics. For the MSE and RMSE metrics, the TA-DGNN model with GCN layers achieves the lowest values. On the other hand, the TA-DGNN model with SAGE layers performs better in terms of MAE, MAPE, and MASE metrics.

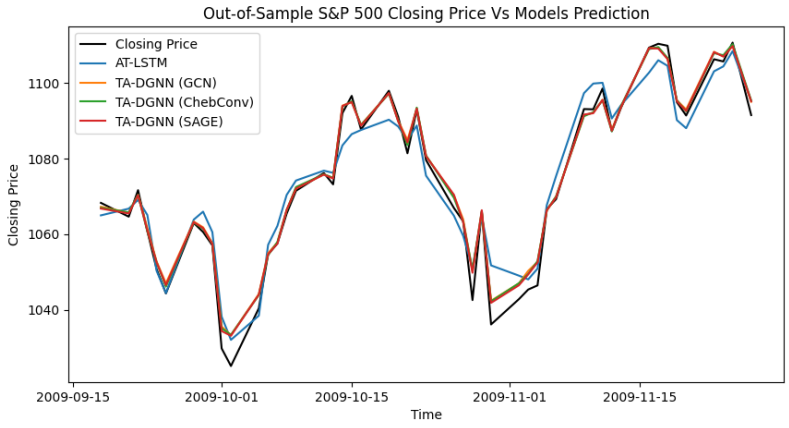

Fig. 4. Comparisons among the models' predictions and the S&P 500 closing price from 19/09/2009 to 27/11/2009.

6 Conclusion

In this research, we propose a model that combines DGNNs with LSTM networks enhanced with an attention mechanism. This integrated model, referred to as TA-DGNN, effectively captures and exploits both the temporal and geometric patterns present in time series data. We evaluate the performance of TA-DGNN in predicting the one-step ahead closing price of the S&P 500 stock index, using the Close, Open, High, and Low prices as features. The evaluation metrics considered are MSE, RMSE, MAE, MAPE, and MASE. Our results demonstrate that TA-DGNN outperforms the baseline models in all the evaluation metrics. Specifically, when considering the GCN layers inside TA-DGNN, we achieve the best results on the test set with values of 123.4954, 11.1128, 7.6060, 0.1881%, and 0.2136 for MSE, RMSE, MAE, MAPE, and MASE, respectively. To assess the robustness and stability of TA-DGNN, we also conduct an out-of-sample analysis. In this analysis, TA-DGNN consistently outperforms the baseline models. The presence of the GCN layers yields the best results for MSE and RMSE with values of 5.5232 and 2.3501, respectively, while the presence of the SAGE layers yields the best results for MAE, MAPE, and MASE with values of 1.7336, 0.1710%, and 0.1926, respectively.

For future research, it is recommended to explore additional features or modify the existing features to further enhance the prediction performance of the model: e.g., the number of past observations (m) and the number of future predicted values (q) could be further investigated and optimized. Another avenue to consider is incorporating mechanisms to track and update the hidden layers within DGNNs, enabling a more efficient capture of temporal and geometric patterns. Additionally, researchers could investigate the performance of TA-DGNN within a trading strategy context to assess its effectiveness in practical applications.

References

1. Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M.: Time series analysis: forecasting and control. 5th edn. **John Wiley & Sons**. (2015)
2. Lamberton, D., & Lapeyre, B.: Introduction to Stochastic Calculus Applied to Finance. 2nd edn. **Chapman & Hall**. (2007)
3. Tankov, P., & Cont, R.: Financial Modelling with Jump Processes. 1st edn. **Chapman & Hall**. (2003)
4. Elman, J. L.: Finding structure in time. **Cognitive Science**, 14(2), 179-211. (1990)
5. Lacasa, L., Luque, B., Ballesteros, F., Luque, J., & Nuno, J. C.: From time series to complex networks: the visibility graph. **Proceedings of the National Academy of Sciences**, 105(13), 4972-4975. (2008)
6. Mandelbrot, B. B., Gomory, R. E., & Cootner, P. H.: Fractals and Scaling in Finance: Discontinuity, Concentration, Risk. New York (N.Y.): **Springer**. (1997)
7. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G.: The graph neural network model. **IEEE Transactions on Neural Networks**, 20(1), 61-80. (2008)
8. Hyndman, R. J., & Koehler, A. B: Another look at measures of forecast accuracy. **International Journal of Forecasting**, 22(4), 679-688. (2006).
9. Fama, E. F.: Efficient capital markets: a review of theory and empirical work. **The Journal of Finance**, 25(2), 383–417. https://doi.org/10.2307/2325486. (1970)
10. Brockwell, P. J., & Davis, R. A. (Eds.): Introduction to Time Series and Forecasting. New York (N.Y.): **Springer**. (2002)
11. Tsay, R. S.: Analysis of Financial Time Series. 2nd edn. **John Wiley & Sons**. (2005)
12. Ariyo, A. A., Adewumi, A. O., &Ayo, C. K.: Stock price prediction using the ARIMA model. In: **2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation** (pp. 106-112). IEEE. (2014, March)
13. Dhyani, B., Kumar, M., Verma, P., & Jain, A.: Stock market forecasting technique using ARIMA model. **International Journal of Recent Technology and Engineering**, 8(6), 2694-2697. (2020)
14. Leung, M. T., Daouk, H., & Chen, A. S.: Forecasting stock indices: a comparison of classification and level estimation models. **International Journal of forecasting**, 16(2), 173-190. (2000)
15. Lee, S. K., Nguyen, L. T., & Sy, M. O.: Comparative study of volatility forecasting models: the case of Malaysia, Indonesia, Hong Kong and Japan stock markets. **Economics**, 5(4), 299-310. (2017)
16. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, **MIT Press**, http://www.deeplearningbook.org. (2016)
17. Zhang, A., Lipton, Z. C., Li, M. & Smola, A. J.: Dive into deep learning, **arXiv preprint arXiv:2106.11342**, https://d2l.ai/. (2021)
18. Selvin, S., Vinayakumar, R., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. P.: Stock price prediction using LSTM, RNN and CNN-sliding window model. In: **2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)** (pp. 1643-1647). IEEE. (2017, September)
19. Roondiwala, M., Patel, H., & Varma, S.: Predicting stock prices using LSTM. **International Journal of Science and Research (IJSR)**, 6(4), 1754-1756. (2017)
20. Zhang, L., Aggarwal, C., Qi, & G. J.: Stock price prediction via discovering multi-frequency trading patterns. In: **Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining** (pp. 2141-2149). (2017)

21. Ding, Q., Wu, S., Sun, H., Guo, J., & Guo, J.: Hierarchical multi-scale Gaussian transformer for stock movement prediction. In **IJCAI** (pp. 4640-4646). (2020)
22. Lu, W., Li, J., Wang, J., & Qin, L.: A CNN-BiLSTM-AM method for stock price prediction. **Neural Computing and Applications**, 33, 4741-4753. (2021)
23. Lin, Y., Huang, Q., Zhong, Q., Li, M., Li, Y., & Ma, F.: A new attention-based LSTM model for closing stock price prediction. **International Journal of Financial Engineering**, 09. https://doi.org/10.1142/S2424786322500141 (2022)
24. Chen, Y., Wei, Z., & Huang, X.: Incorporating corporation relationship via graph convolutional neural networks for stock price prediction. In: **Proceedings of the 27th ACM International Conference on Information and Knowledge Management** (pp. 1655-1658). (2018)
25. Matsunaga, D., Suzumura, T., 1& Takahashi, T.: Exploring graph neural networks for stock market predictions with rolling window analysis. **arXiv preprint arXiv:1909.10660**. (2019)
26. Lazcano, A., Herrera, P. J., & Monge, M.: A combined model based on recurrent neural networks and graph convolutional networks for financial time series forecasting. **Mathematics**, 11(1), 224. (2023)
27. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y.: A comprehensive survey on graph neural networks. **IEEE Transactions on Neural Networks and Learning Systems**, 32(1), 4-24. (2020)
28. Kipf, T. N., & Welling, M.: Semi-supervised classification with graph convolutional networks. In: **arXiv preprint arXiv:1609.02907**. (2016)
29. Hamilton, W., Ying, Z., & Leskovec, J.; Inductive representation learning on large graphs. In: **Advances in Neural Information Processing Systems**, 30. (2017)
30. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y.: Graph attention networks. **arXiv preprint arXiv:1710.10903.**. (2018)
31. Skarding, J., Gabrys, B., & Musial, K.: Foundations and modeling of dynamic networks using dynamic graph neural networks: a survey. **IEEE Access**, 9, 79143-79168. (2021)
32. Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., & Li, H: T-GCN: A temporal graph convolutional network for traffic prediction. **IEEE Transactions on Intelligent Transportation Systems**, 21(9), 3848-3858. (2019)
33. Wu, Z., Pan, S., Long, G., Jiang, J., & Zhang, C.: Graph wavenet for deep spatial-temporal graph modeling. **arXiv preprint arXiv:1906.00121**. (2019)
34. Ruiz, L., Gama, F., & Ribeiro, A.: Gated graph recurrent neural networks. **IEEE Transactions on Signal Processing**, 68, 6303-6318. (2020)
35. Wang, Y., Yuan, Y., Ma, Y., & Wang, G.: Time-dependent graphs: definitions, applications, and algorithms. **Data Science and Engineering**, 4, 352-366. (2019)
36. Time series to visibility graphs (ts2vg) python packages, https://cbergillos.com/ts2vg
37. Hochreiter, & S.,Schmidhuber, J.: Long short-term memory. **Neural Computation**, 9(8), 1735-1780. (1997)
38. Bahdanau, D., Cho, K., & Bengio, Y.: Neural machine translation by jointly learning to align and translate. **arXiv preprint arXiv:1409.0473**. (2014)
39. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Polosukhin, I.: Attention is all you need. In **Advances in Neural Information Processing Systems**, 30. (2017)
40. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E.: Neural message passing for quantum chemistry. In **International Conference on Machine Learning** (pp. 1263-1272). PMLR. (2017, July)

41. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K. I., & Jegelka, S.: Representation learning on graphs with jumping knowledge networks. **In International conference on machine learning** (pp. 5453-5462). PMLR. (2018, July)
42. You, J., Du, T., & Leskovec, J.: ROLAND: graph learning framework for dynamic graphs. In **Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining** (pp. 2358-2366). (2022, August)
43. Defferrard, M., Bresson, X., & Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: **Advances in Neural Information Processing Systems**, 29. (2016)
44. Hahnloser, R., Sarpeshkar, R., Mahowald, M. et al.: Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. **Nature** 405, 947–951. https://doi.org/10.1038/35016072 (2000)
45. Chordia, T., Roll, R., & Subrahmanyam, A.: Market liquidity and trading activity. **The Journal of Finance**, 56(2), 501-530. (2001)
46. Amihud, Y.: Illiquidity and stock returns: cross-section and time-series effects. **Journal of Financial Markets**, 5(1), 31-56. (2002)
47. S&P Global Homepage, https://www.spglobal.com
48. Welch, I., & Goyal, A.: A comprehensive look at the empirical performance of equity premium prediction. **The Review of Financial Studies**, 21(4), 1455-1508. (2008)
49. Yahoo Finance Homepage, https://finance.yahoo.com
50. Malkiel, B. G.: Efficient market hypothesis. **Finance**, 127-134. (1989)
51. Kingma, D. P., & Ba, J.: Adam: a method for stochastic optimization. **arXiv preprint arXiv:1412.6980**. (2014)
52. Akaike, H.: Information theory and an extension of the maximum likelihood principle. **Selected Papers of Hirotugu Akaike**, 199-213. (1998)