# Evaluation of Deep Learning and Traditional Models for Index-Level and Stock-Level Financial Forecasting

**Tao Yu**

**Abstract** In financial market forecasting, numerous methods have been proposed for predicting both stock indices and individual stock prices. However, systematic evaluations comparing these models across different prediction tasks remain limited. To address this gap, this study conducts a unified comparative analysis of three deep learning models (MLP, LSTM, and Transformer) and two traditional benchmark models (ARIMA and SVR). All models are evaluated under a consistent experimental framework using the same dataset, input window lengths, and prediction horizons, and are tested using a rolling forecasting mechanism. Model performance is assessed using mean absolute error (MAE), root mean square error (RMSE), and prediction accuracy based on relative error thresholds. The results show that: (1) performance differences among forecasting models are largely determined by their structural characteristics, and conclusions derived from index-level forecasting cannot be directly generalized to stock-level prediction tasks. (2) Among deep learning models, LSTM demonstrates the most stable overall performance across different prediction settings, as its gated recurrent structure enables robust modeling of temporal dependencies in non-stationary financial time series. Transformer is more effective in capturing long-term trends but is less capable of modeling local fluctuations under high volatility, while MLP shows strong sensitivity to prediction configurations such as input window length and forecasting horizon. (3) Among traditional models, ARIMA exhibits strong stability in short-term index forecasting and maintains relatively low prediction errors across different settings, whereas SVR experiences significant performance degradation when longer historical input windows are used, highlighting the limitations of static kernel-based regression in modeling high-dimensional temporal features. (4) At the stock level, the higher heterogeneity of individual stock price series further amplifies performance differences among models. For example, on NVIDIA stock, most models achieve prediction accuracies below 50% under the ±10% tolerance, indicating that stock-level forecasting requires models tailored to specific prediction targets.

**Keywords** Financial Time Series Forecasting, Stock Index Prediction, Stock Price Prediction, Deep Learning Models, Traditional Forecasting Models (6)Benchmark Evaluation

## 1 Introduction

In today's closely connected global financial markets, investors and researchers have become more dependent on stock indices for understanding the overall trend of the market, shifts in industrial structure, and aggregated risk sentiment[1]. Among them, the Nasdaq index is a significant market benchmark that represents high-tech and growth-oriented enterprises, and it is especially responsive to macroeconomic changes, fluctuations in market sentiment, and alterations in the technology cycle. Due to its broad composition and relatively concentrated weight structure, the price changes of the Nasdaq index are mainly influenced by systematic factors such as the macroeconomic environment, market psychology and sector rotation. As a result, the Nasdaq index has become not only an essential benchmark for global capital markets but also a crucial reference standard for judging investors' risk preferences and technology cycle trends. Therefore, predicting the Nasdaq Index can provide reference information for

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

monitoring the market, making asset allocation plans and risk-hedging strategies, as well as conducting theoretical research on issues such as market efficiency, financial cycle changes, and asset price determination mechanisms. Also, it differs from single stock prediction in that index prediction focuses on systematic macroeconomics and industry environment changes at the market level instead of individual firm events or finances. Because we can diversify our idiosyncratic risks at the index level[2], price dynamics are generally smoother[3], so index forecasting is more appropriate for asset allocation and risk management; single stock forecasts need to take into account firm fundamentals, information disclosure, events etc., they usually have larger volatilities and uncertainties[4,5].

To predict the movement of Nasdaq index and individual stocks, many different methods have been proposed over time, which can roughly be divided into several categories as follows: (1) classical statistical and econometric approaches, where their main idea is based on assuming that financial time series follow some kind of statistical structure. These methods have a good theoretical basis and can be easily understood, so they are suitable for predicting indexes, but not very suitable for predicting highly fluctuating individual stocks. Some representative models include the Autoregressive (AR) model and the Autoregressive Integrated Moving Average (ARIMA) model[6]. (2) *Feature-based traditional machine learning* approach that converts the time-series prediction problem into either a regression or classification problem after manually crafting various technical indicators and historical features. The former works well for index prediction but depends much on the quality of feature design when it comes to predicting an individual stock. The typical example is support vector regression (SVR)[7,8] and random forest models[9]. (3) *Deep learning method* is that can automatically learn the complex non-linear pattern and temporal correlation among past price series through the use of neural network, it has also been proven with good forecasting performance on both index level as well as individual stock prediction. Among which MLPs model non-linear mappings using feed-forward architecture[10], recurrent neural network models such as long short-term memory (LSTM) capture long-term temporal dependencies via gating mechanisms[11,12], and attention-based transformer models can model long-range dependency well and make it more computationally parallel[13,14].

Though many techniques have been devised to predict financial time series, ranging from statistical models to traditional machine learning algorithms as well as various kinds of deep learning architecture, systematic comparisons of model performances on different methods under a common set-up still remain quite limited in the existing literature. Especially with respect to predictive stability, applicability and sensitivity to forecasting configurations when using the same datasets, forecasting tasks and evaluation metrics, there is still no sufficient evidence for model differences. In real-world markets[15], key settings such as input window length and forecasting horizon are often adjusted according to strategy requirements and changing market conditions. If a model is highly sensitive to such settings, it will be prone to producing unstable forecast signals, thereby reducing the robustness of asset allocation, risk management and other aspects, as well as trading decisions. In addition, due to the different structures of volatility and heterogeneity among index-level and stock-level series[16], the lack of unified comparisons makes it difficult to determine the transferability of empirical conclusions, which may further limit the practical value of forecasting models across different prediction targets. Therefore, to improve the reproductivity and interpretability of research results and provide more reliable empirical support for model selection and strategy application in index-level and stock-level forecast tasks.

In view of the above research deficiencies, this paper systematically benchmarks the performance of typical prediction models in different areas through an overall comparison. Specifically, this paper selects three typical deep learning models - LSTM, MLP, and Transformer; two non-deep-learning baselines, the classical statistical model ARIMA and the traditional machine learning model SVR; And use identical assessment indicators to compare these five prediction methods. The experiment includes two levels of hierarchical prediction tasks: the Nasdaq Composite Index (at the index level) and six typical individual stocks (at the stock level), aiming to explore whether the model has generalised ability for various objects being predicted. The results show that there is an obvious difference in predictive stability and response degree to forecast configuration (such as input window length and forecasting horizon)). In terms of index-level prediction, LSTM has

achieved the most stable performance under different windows and horizons. The Transformer has shown some limitations when dealing with the overall trend of short-term forecasting in a multi-step scenario, and its performance is more unstable. MLP still performs well under some settings, but it is more sensitive to the experiment environment than other models. As for the traditional models, ARIMA can be said to give a fairly solid baseline for short term prediction and SVR is less good on average. Stock level, in terms of predicting, the performance difference gets much bigger; for those very volatile or nearly non-stationary stocks like NVIDIA, most models suffer greatly in their performance, which suggests it is becoming more noticeable that there is a greater difficulty with increasing complexity and heterogeneity at stock-level prediction. Generally speaking, whether or not the prediction results are good depends mainly on the hierarchy and feature characteristics of what we predict at that moment; therefore, it is impossible to draw conclusions from indexes. The above findings can be used as a reference in terms of which model to use and deploy for financial time-series forecasting.

## 2 Methodology

As there is no systematic evaluation of forecasting approaches on both stock indices and individual stocks in the existing studies, this paper attempts to comprehensively evaluate the predictive power of some representative deep learning models at both the index level and the single stock level. Also include 1 representative model from the classical statistical/econometric models and 1 representative model from the traditional feature-based machine learning method for comparison with deep learning models. In particular, the study will first compare how well various models performed when predicting an index price compared to a single stock price. Then it explores if the temporal dependency structure and prediction behavior that are learned by those models still hold true for index-level as well as stock-level time series[17].

### 2.1 Comparison Methods

In order to systematically evaluate the performance of different forecast approaches at both the index level and the individual stock level in the same experimental setting for financial time-series prediction problems, we choose three state-of-the-art deep learning models as follows: MLP (Multilayer perceptron), LSTM(Long short-term memory), Transformer. Also, 2 non-deep learning baselines: the classical stat model ARIMA and the tradition mach-learn model SVR, for more comprehensive & representative comparisons. In the following subsections we will present all mathematical formulations, parameters and concrete usage of models for financial time-series forecasting.

#### 2.1.1 MLP

**(1) Mathematical formulation**

Given an input feature vector $\mathbf{x} \in R^d$, the MLP performs a series of non-linear transformations that can be described as:

$$\mathbf{h}^{(l)} = \phi\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right), \tag{1}$$

where $\mathbf{h}^{(0)} = \mathbf{x}$, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ denote the weight matrix and bias vector of the $l$-th layer, respectively, and $\phi(\cdot)$ represents the ReLU activation function. For regression tasks, the output layer adopts a linear mapping to produce the predicted value $\hat{y}$ .

**(2) Parameter settings in this study**

The multilayer perceptron (MLP) in this study has a fully connected feedforward structure and works with a flattened input feature vector. The network is composed of three hidden layers with decreasing unit quantities, namely 256, 128, and 64. Each hidden layer employs the ReLU activation function and incorporates L2 regularization with a coefficient of $1 \times 10^{-4}$ to mitigate overfitting during training. To further enhance training stability, batch normalization is applied after each hidden layer. In addition, dropout with a rate of 0.3 is introduced after the first two hidden layers to improve robustness against noise. Finally, the model produces a single scalar output through a linear regression head, representing the predicted price value. In terms of model size, the MLP contains 45,825 parameters in total, including 44,929 trainable parameters and 896 non-trainable parameters (from batch normalization running statistics). The dominant computational cost of one forward pass is due to dense matrix multiplications, which scales as $\mathcal{O}(d \cdot 256 + 256 \cdot 128 + 128 \cdot 64 + 64 \cdot 1)$, $d$ denotes the dimensionality of the flattened input feature vector.

**(3) MLP for stock and index prediction**

In stock price and index forecasting tasks, since the multilayer perceptron does not explicitly model temporal dependencies, a sliding window strategy is adopted to

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

construct input features from historical price series. Specifically, a fixed-length window of past observations is flattened into an input vector. The MLP then maps this input vector to a single scalar output through a series of nonlinear transformations, corresponding to the predicted index price at a future horizon $h$. In this manner, historical information from the time series is implicitly encoded into the input features[18], enabling the MLP to perform stock price and index prediction.

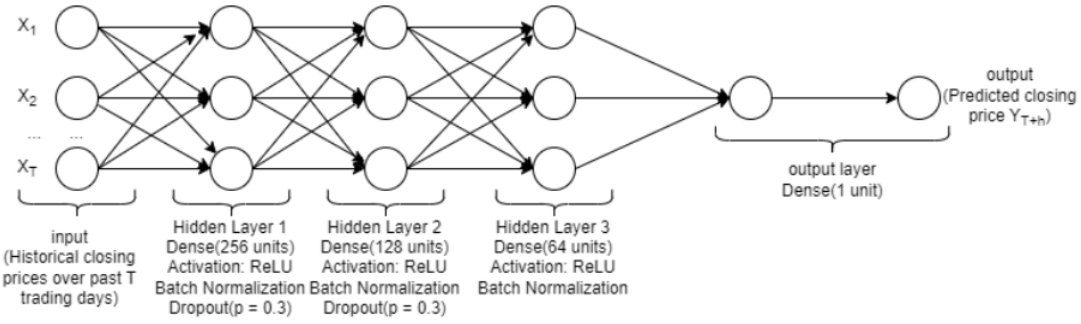



**Figure 1. Architecture of the multilayer perceptron (MLP) model used in this study.**

### 2.1.2 LSTM

**(1) Mathematical formulation**

Given an input sequence $\left\{\mathbf{x}^{t}\right\}_{t=1}^{T}$, the Long Short-Term Memory (LSTM) network models temporal dependencies through a gated recurrent mechanism. At each time step $t$, the LSTM updates its hidden state according to

$$\begin{aligned}
&\mathbf{i}_t = \sigma\left(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_t - 1 + \mathbf{b}_i\right), \\
&\mathbf{f}t = \sigma\left(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f\right), \\
&\mathbf{o}_t = \sigma\left(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o\right), \\
&\tilde{\boldsymbol{c}}_t = \tanh\left(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c\right), \\
&\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\boldsymbol{c}}_t, \\
&\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned} \tag{2}$$

where $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ denote the input, forget, and output gates, respectively, $\mathbf{c}_t$ represents the memory cell state, and $\mathbf{h}_t$ is the hidden state at time step $t$.

**(2) Parameter settings in this study**

The LSTM model in this study is composed of two stacked LSTM hidden layers. The first LSTM layer contains 64 hidden units to extract high-dimensional temporal features from the input sequence, while the second layer contains 32 hidden units to further compress and integrate temporal information. In terms of the inter-layer architecture, the output of the first LSTM layer is propagated to the subsequent recurrent layer in the form of a full sequence, ensuring that temporal dependencies across time steps are effectively captured. To avoid overfitting of the model during training, a drop-out regularisation term is introduced after each LSTM hidden layer with a drop-out rate of 0.3. Finally, the temporal representation learned by the recurrent network is fed into a fully connected regression output layer, producing a single scalar prediction corresponding to the closing price.

In terms of model size, the LSTM model contains 29,345 parameters in total, all of which are trainable (trainable parameters = 29,345). Specifically, the first LSTM layer has 16,896 parameters, the second LSTM layer has 12,416 parameters, and the output layer Dense(1) has 33 parameters. Moreover, given the input shape ($T$, 1), the input feature dimension is $d$ = 1, indicating that only a univariate sequence is used at each time step; the first-layer output shape (10, 64) implies that the input window length is $T$ = 10. Regarding computational complexity, the dominant cost of LSTM arises from gate computations and matrix multiplications at each time step. For an LSTM layer with hidden size $h$, input dimension $d$, and sequence length $T$, the time complexity of a single forward pass can be approximated as $\mathcal{O}\left(T \cdot 4(dh + h2)\right)$ .Therefore, the overall computational complexity of the two-layer stacked LSTM in this study can be expressed as $\mathcal{O}\left(T \cdot 4(d \cdot 64 + 642) + T \cdot 4(64 \cdot 32 + 322)\right)$.

**(3) LSTM for stock and index prediction**

In stock price and index forecasting tasks, the LSTM model takes a sequence of historical index prices over the past $T$ trading days as input. Through a gated recurrent mechanism, the model updates its hidden states along the temporal dimension, enabling it to capture both short-

term fluctuations and long-term trends in the price series. Stacked LSTM hidden layers further abstract and integrate temporal features, and the learned temporal representations are finally fed into a fully connected regression output layer to generate predictions of stock prices or index values[19].

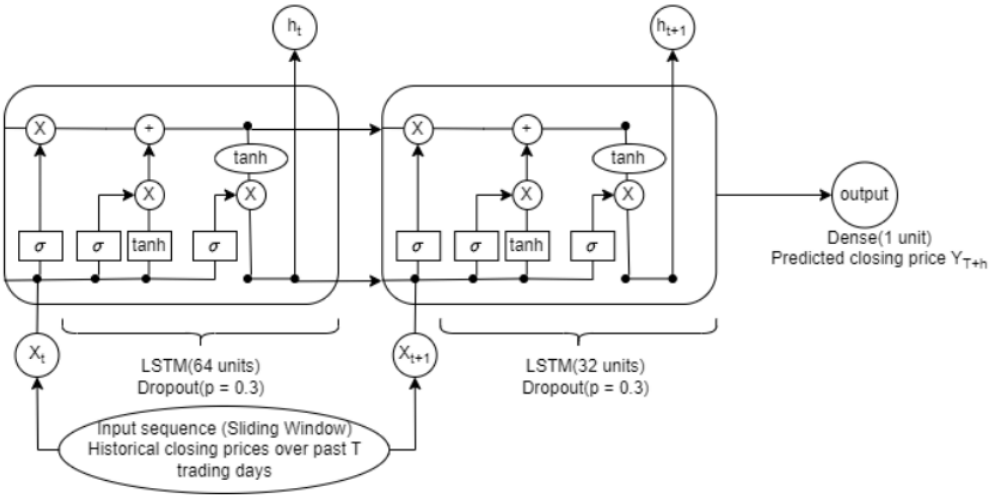



**Figure 2. Architecture of the Long Short-Term Memory (LSTM) model used in this study.**

**2.1.3 Transformer**

**(1) Mathematical formulation**

Given an input sequence $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T] \in R^{T \times d}$, the Transformer encoder first projects the input into a *d*-dimensional latent space and incorporates positional information through a positional encoding. The core operation of each encoder block is based on multi-head self-attention, defined as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \qquad (3)$$

where **Q**, **K**, and **V** denote the query, key, and value matrices, respectively, and *dk* is the dimensionality of the key vectors.

**(2) Parameter settings in this study**

In this study, the Transformer model adopts a pure encoder-only architecture to capture temporal dependencies within the input time-series window[20]. First, the univariate input sequence is linearly projected into a latent representation with a dimensionality of 128. A fixed sinusoidal positional encoding with the same sequence length and dimensionality is then added to inject temporal order information.

The core of the model consists of four stacked Transformer encoder blocks. Each encoder block follows a pre-normalization design and sequentially comprises a multi-head self-attention (MHSA) module and a position-wise feed-forward network (FFN). The MHSA mechanism employs 8 attention heads, with a key dimension of 16 for each head. The FFN is made up of two fully connected layers, each containing 256 and 128 hidden units, respectively; it employs a ReLU activation function within the intermediate layer to enhance its nonlinearity. To enhance the stability of training, residual connections and layer normalisation are introduced into both the self-attention and feed-forward sub-layers. To reduce the risk of overfitting, a dropout regularisation with a rate of 0.1 is applied to each encoder block. After the stack of encoder blocks, a final layer of normalisation is applied to the output representations. That is, in place of global pooling, select the hidden state at the final time step as an overall representation of the entire sequence[21]. Subsequently, this representation is processed by a drop-out layer (rate=0.1), and then it is sent to a linear regression output layer that outputs a single scalar prediction corresponding to the closing price. In terms of model size, this Transformer has a total of 530,561 parameters, and all of them are trainable (trainable parameters = 530,561). The specific values are as follows: There are 256 parameters in the input projection layer (Dense). Within each encoder block, the multi-head self-attention layer has 66,048 parameters. The feed-forward network consists of two fully connected layers: Dense(256) with 33,024 parameters and Dense(128) with 32,896 parameters. In addition, each Layer Normalization layer introduces 256 parameters for scaling and

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

shifting. The regression output layer Dense(1) at the tail of the model contains 129 parameters. In general, we use stacked self-attentions and feed-forward transformations to learn from the high-dimensional temporal space, which requires a much bigger parametric budget compared to classic statistical baselines as well as light-weight neural networks[22].

Regarding computational complexity, the dominant cost of the Transformer arises from self-attention and feed-forward operations. For an input window of length T, the computational complexity of self-attention is approximately $\mathcal{O}(T^2 \cdot d)$, where d denotes the hidden dimension (in this study, $d = 128$), reflecting the quadratic growth with respect to the sequence length. Meanwhile, the complexity of the feed-forward network is $\mathcal{O}(T \cdot d \cdot d_{\text{ff}})$, where $d_{\text{ff}} = 256$. Therefore, after stacking $L = 4$ encoder blocks, the overall computational complexity of the Transformer can be expressed as $\mathcal{O}(L \cdot (T^2 \cdot d + T \cdot d \cdot d_{\text{ff}}))$.

**(3) Transformer for stock and index prediction**

In stock price and index forecasting tasks, the Transformer model takes a sequence of historical index prices over the past $h$ trading days as input. The one-dimensional price series is first linearly projected into a high-dimensional latent representation, and positional encoding is incorporated to preserve temporal order information. Subsequently, stacked Transformer encoder modules model the correlations among different time steps through multi-head self-attention and feed-forward networks, enabling the capture of long-term dependencies in the price series. Finally, the hidden representation corresponding to the final time step is selected as a summary representation of the entire sequence and fed into a linear regression output layer to generate predictions of stock prices or index values.

### 2.1.4 ARIMA

**(1) Mathematical formulation**

Given a univariate time series $\{y_t\}_{t=1}^{T}$, the autoregressive integrated moving average (ARIMA) model transforms a non-stationary series into a stationary one through differencing and performs linear modeling on the transformed series[23]. An ARIMA($p$, $d$, $q$) model first applies d-th order differencing to the original series, which is defined as

$$z_t = \nabla^d y_t = (1 - B)^d y_t , \tag{4}$$

where $B$ denotes the backshift operator such that $B y_t = y_t - 1$.

Based on the stationary differenced series $z_t$, the ARIMA($p$, $d$, $q$) model represents the current observation as a linear combination of its past values and past stochastic disturbances:

$$Z_t = \sum_{i=1}^{p} \varphi_i z_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}, \tag{5}$$

where $p$ and $q$ denote the orders of the autoregressive and moving average components, respectively, $\phi_i$ and $\theta_j$ are model parameters, and $\varepsilon_t$ is a white-noise error term with zero mean and constant variance.

For forecasting, the model recursively predicts the differenced series and then applies inverse differencing to recover predictions on the original scale.

**(2) Parameter settings in this study**

Under a sliding forecast system in this paper, an ARIMA model is built using a fixed-length historical observation window at each forecasting stage to model the time series and generate future price predictions for a specific forecast horizon. In terms of model construction, an ARIMA(1,1,1) configuration is used, and the autoregressive coefficient p=1, the difference order d=1, and the moving average order q=1 are specified. This setting is to make the data more stable through the first-order difference and it has captured a linear dynamic of price changes at lower order autoregressive or moving average components[24]. There is no seasonality added and the seasonal index is (0, 0, 0, 0), so as not to introduce extra periodicity into short-term prediction tasks[3]. There is also a constant term to consider the overall trend of changes in the differenced series. Forecasting employs the stepwise rolling technique of an ARIMA model. That is, at each prediction step, the model parameters are updated based on the past information in the current predicting window, and a forecast for the target period will be produced next. At this time, the new observed true value needs to be added to our history series, and then the forecasting window will be refreshed. This way, a dynamic prediction can continue to change over time.

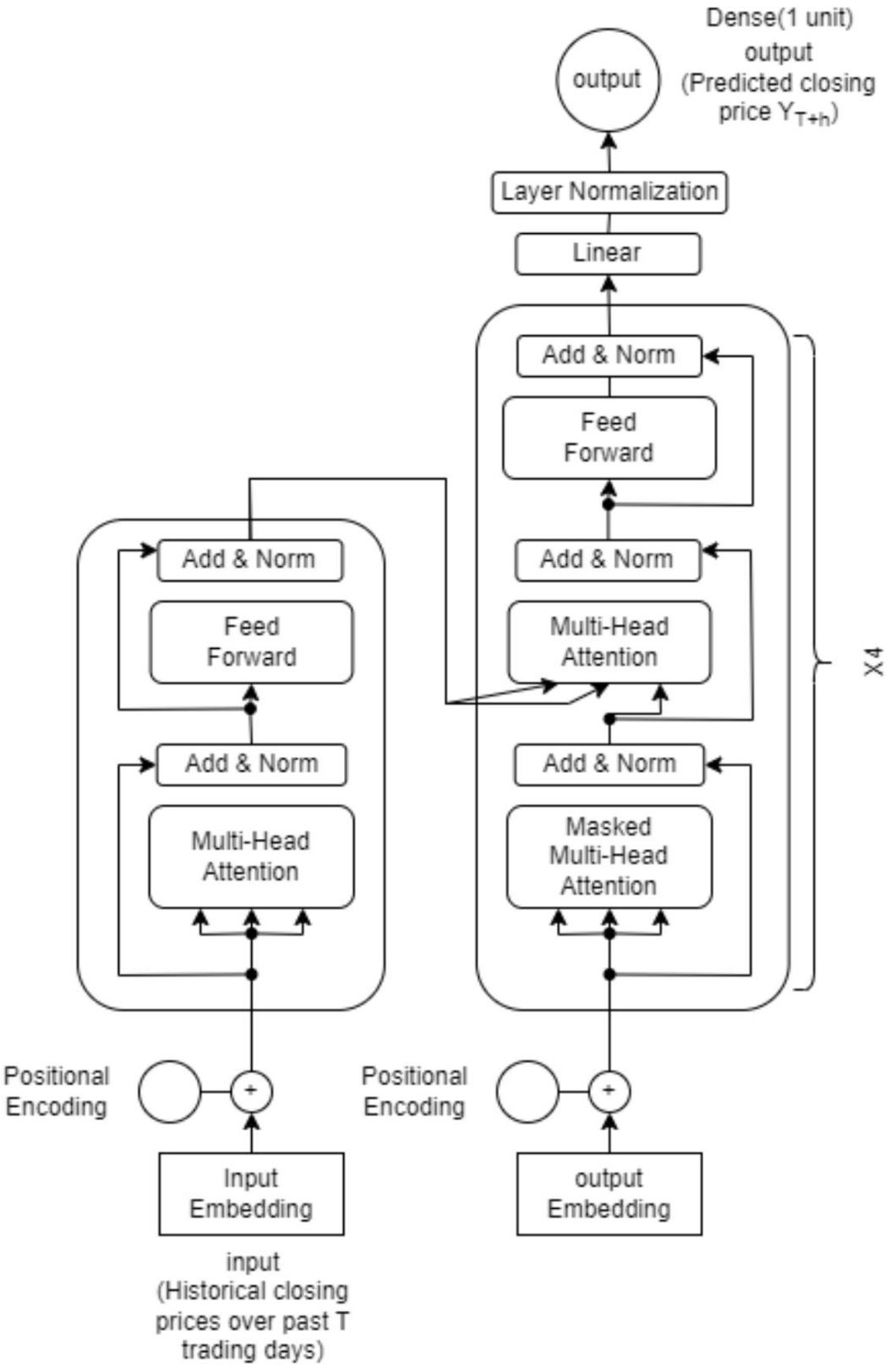



**Figure 3. Architecture of the Transformer model used in this study.**

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

In terms of model size, ARIMA(1,1,1) involves a small number of estimable parameters and offers strong interpretability. Specifically, the model contains one autoregressive coefficient (AR coefficient $\phi_1$), one moving average coefficient (MA coefficient $\theta_1$), one constant/drift term (trend "c"), and one innovation variance parameter ($\sigma^2$ ). Thus, the total number of core parameters to be optimised is about 4, much less than that in deep-learning models.

From a computational perspective, the primary expenses in ARIMA include re-estimating parameters repeatedly when updating each rolling forecast. A history sequence of length T can usually be modelled by maximum likelihood estimation under a state-space setting, and its computation cost is thus an increasing function of T. Since this study adopts a walk-forward protocol and refits the model at every time step in the test phase, the overall computational complexity can be expressed as $\mathcal{O}(N_{\text{test}} \cdot C_{\text{fit}}(T))$, where $N_{\text{test}}$ denotes the number of rolling forecasting steps during testing, and $C_{\text{fit}}(T)$ represents the fitting cost under a window length of *T*.

**(3) ARIMA for stock and index prediction**

For the stock and index price forecast problem, using the ARIMA model to obtain a point forecast for future prices based on the linear dynamic relationship between historical price time series. In terms of prediction, as shown in Figure 3.2: Take the price series as a single variable time-series input, and inside a sliding forecast, it is used as the target to determine whether the next prediction result will be updated. In each forecasting stage, only the historical price data within the current prediction window are used to estimate the model parameters, and subsequent prices beyond this period are predicted for the next forecast. The most recent observation of the true value will be added to the historical data sequence to extend the forecast time window and continue prediction. This kind of rolling prediction mechanism can simulate an online forecasting scenario in reality, and it does not need to introduce future information at all.

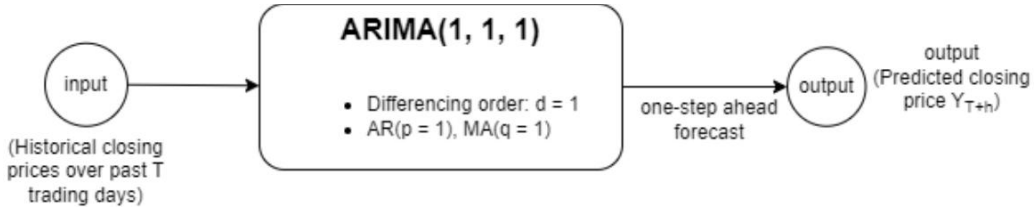



**Figure 4. Architecture of the ARIMA model used in this study.**

**2.1.5 SVR**

**(1) Mathematical formulation**

Support Vector Regression (SVR) is a supervised learning method based on the principle of structural risk minimization[7], which aims to learn a mapping between input features and a continuous target variable. Given a training dataset $\{(x_i, y_i)\}_{i=1}^{N}$,, where $\mathbf{x}_i \in R^d$ denotes the input feature vector of the *i*-th sample and $y_i \in R$ denotes the corresponding target value, SVR constructs the regression function by solving the following optimization problem:

$$\min_{w,b,\xi_i,\xi_i^*} \frac{1}{2}\| |w| \|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*), \tag{6}$$

subject to the constraints

$$\begin{aligned} &y_i - (\mathbf{w}^T\phi(\mathbf{x}i) + b) \leqslant \varepsilon + \xi_i , \\ &(\mathbf{w}^T\phi(\mathbf{x}i) + b) - yi \leqslant \varepsilon + \xi_i , \\ &\xi_i , \ \xi_i^* \ \geqslant 0, \end{aligned} \tag{7}$$

where $\phi(\cdot)$ is a nonlinear transformation mapping input data into a high-dimensional feature space, and ε represents the width of the ε-insensitive loss function.

The resulting prediction function is given by[8]

$$\hat{y} = f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + b. \tag{8}$$

In stock and index price forecasting tasks, **x***i* typically consists of features constructed from historical price observations within a forecasting window, while *yi* represents the future price at the specified forecast horizon. The parameters **w** and *b* determine the form of the regression function; $\phi(\cdot)$ implicitly models nonlinear relationships through kernel functions; $\varepsilon$ specifies the tolerance margin for prediction errors; and are slack variables that measure deviations outside the $\varepsilon$-insensitive zone; and the regularization parameter *C* controls the trade-off between model complexity and fitting error.

**(2) Parameter settings in this study**

Based on the SVR with a RBF kernel in our research to explore which kinds of non-linear mapping relationships exist between past price information and current or future prices[25]. Regularization item C = 100, which regulates

the equilibrium between model complexity and predictive error. Set the width of the ϵ-insensitive loss region to 0.01, thereby allowing the model to be more tolerant of some small prediction errors that will make our regression results more robust. The kernel scale parameter is adaptively set as γ=scale so that the kernel function can adjust its effective range depending on how spread out the input feature values are.

Since SVR models cannot represent temporal dependency explicitly, for the input construction, all historical price information falling within the forecast window is flattened and handed over as a 1D feature vector to be processed by the model. In the prediction stage, after completing parameter estimation for the training data by a sliding-window method to establish the SVR model. Then make predictions based on this trained SVR model for test samples. To ensure that we evaluate our predicted values at the same scale as the original price series, we need to perform an inverse normalisation operation and restore the predictions to their original prices.

From a model size perspective, it cannot demonstrate that the SVR has achieved full-parametrization effect after setting up a certain number of layers or having an explicit weight matrix; rather, support vectors at training time significantly influence this outcome. The function for prediction:

$$f(\mathbf{x}) = \sum_{i=1}^{N_{SV}} \left(\alpha_i + \alpha_i^*\right) K(\mathbf{x}_i, \mathbf{x}) + b, \tag{9}$$

where $N_{SV}$ denotes the number of support vectors, $\alpha i$ and $\alpha i*$ are the corresponding coefficients, $b$ is the bias term, and $K(\cdot, \cdot)$ represents the kernel function. Therefore, the model complexity of SVR mainly increases with $N_{SV}$: a larger number of support vectors generally yields stronger representational capacity, but also incurs higher computational and memory costs.

Regarding computational complexity, the dominant inference cost of SVR comes from kernel evaluations between test samples and the support vectors. Since each prediction requires similarity computations with all support vectors, the prediction complexity can be expressed as: $\mathcal{O}(N_{SV} \cdot d)$, where $N_{SV}$ is the number of support vectors and $d$ is the input feature dimension (in this study, $d$ = 10 after flattening the input window). Consequently, as $NSV$ increases, the computational cost of SVR inference rises accordingly.

**(3) SVR for stock and index prediction**

In stock and index price forecasting tasks, the support vector regression (SVR) model performs point prediction of future prices by learning nonlinear mapping relationships between historical prices and future prices in a high-dimensional feature space. Unlike sequence models that explicitly model temporal dependencies, SVR is a static regression approach whose predictive capability primarily relies on the historical information encoded in the input features.

In the forecasting procedure, the SVR model is applied within a sliding forecasting framework. Specifically, historical price observations within the forecasting window are first constructed into fixedlength input feature vectors, thereby transforming the time-series forecasting problem into a standard supervised regression task. At each forecasting step, the model generates price predictions for the specified forecast horizon based on samples within the current forecasting window. The most recently observed true value is then incorporated into the historical sequence to advance the forecasting window and complete the subsequent prediction step. Thus, it is possible to achieve dynamic updates of the forecast plan without involving information from the future.

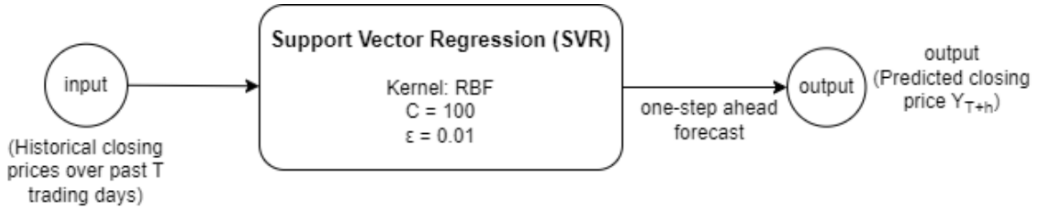



**Figure 5. Architecture of the SVR model used in this study.**

## 2.2 Training Configuration

To ensure a fair comparison[17], all deep learning models in this study (MLP, LSTM, and Transformer) are trained under exactly the same training configuration. Since the forecasting task is formulated as a regression problem,

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

mean squared error (MSE) is adopted as the training objective. Model parameters are optimized using the Adam optimizer with its default learning rate setting ($1 \times 10-3$ ) [26], and mean absolute error (MAE) is additionally reported as an auxiliary evaluation metric[27]. All deep-learning-based models are trained for 100 epochs, and mini-batch gradient descent is used to update parameters; the default batch size in Keras was set at 32. During training, model parameters are adjusted repeatedly based on each mini-batch to achieve a sharp drop in the loss function at the beginning of each epoch followed by slow convergence; thus, it is proven that the system has achieved stable convergence. In summary, all deep-learning-based models use the same training pipeline and hyperparameters to ensure comparability among them.

ARIMA and SVR models have different modelling paradigms at their core than deep learning; therefore, in this paper, we design corresponding training and prediction strategies according to their characteristics. As a classic statistical time series model, ARIMA uses a step-by-step rolling re-estimation method in forecasting; that is, during each prediction step, the model parameters are updated based on the current historical observation window, and then forecasts are made for the specified horizon. The most recent true observation is then added to the historical series, and the forecasting window moves one step forward. This approach is closer to the practical application of statistical models in time series analysis. On the other hand, SVR is a static supervised regression model that learns its parameters only once from the training set and does not change them subsequently. In the prediction phase, SVR produces predictions in sequence using a rolling forecast approach and maintains the model parameters constant during the entire test period[28].

Therefore, although the parameter learning mechanisms of ARIMA, SVR and deep learning models vary, they are all evaluated within the same rolling forecast framework with identical input window sizes and forecasting horizons. This design can ensure the fairness and comparability of different model approaches in terms of task definition and evaluation rules.

# 3 Experiments

## 3.1 Experimental Setup

### 3.1.1 Datasets

**(1) NASDAQ Index dataset**

The dataset employed in this study is obtained from the NASDAQ official website[29] and encompasses daily trading records of the NASDAQ Composite Index from 07/08/2014 to 07/05/ 2024, comprising a total of 2,529 observations. Each record corresponds to a single trading day and contains basic financial indicators, including the opening price, closing price, highest price of the day and lowest price of the day. The Descriptive Statistics of the Daily Closing Prices for the NASDAQ Composite Index are as follows.

**Table 1. Descriptive statistics of the NASDAQ Composite Index closing prices.**

| Variable | Mean | Std | Min | Median | Max |
|---|---|---|---|---|---|
| Close price | 9,204.76 | 4,385.33 | 3,765.28 | 7,629.35 | 20,391.97 |

**(2) Individual stock datasets**

The individual stock datasets employed in this study are obtained from the NASDAQ official website[29] and include six stocks: Amazon, Apple, Meta, Microsoft, NVIDIA, and Tesla. Databases contain daily trading information for all listed companies from October 15, 2015 to August 9, 2025; there were 2,513 records per company. Observations are a single day's trading, containing basic financial indicators such as trade volume, opening price, closing price, highest price and lowest price. Below is a summary statistics table of the closing prices for all 6 individual stocks.

**Table 2. Descriptive statistics of closing prices for the six individual stocks.**

| Stock | Mean | Std | Min | Median | Max |
|---|---|---|---|---|---|
| Amazon | 116.04 | 56.75 | 24.10 | 106.21 | 242.06 |
| Apple | 111.00 | 70.16 | 22.59 | 116.03 | 259.02 |

| Stock | Mean | Std | Min | Median | Max |
|---|---|---|---|---|---|
| Meta | 267.29 | 164.84 | 88.91 | 196.40 | 790.00 |
| Microsoft | 219.14 | 134.81 | 46.68 | 213.02 | 535.64 |
| NVIDIA | 31.82 | 45.95 | 0.63 | 12.54 | 189.11 |
| Tesla | 139.89 | 124.43 | 9.58 | 136.03 | 479.86 |

**(3) Dataset Split**

In order to maintain the time sequence of the dataset and avoid different observation times leaking information, we divide it according to whether they occurred into a training set and a test set; generally speaking, about 70% will be our training set, and the remaining 30% will be our reserved test set.

**3.1.2 Forecasting Task Definition**

**(1) Prediction Target**

This paper selects as the research object: for each company's and market index, predict the closing prices of them for every single day in the future based on the past data[3]. In other words, based on historical observations of prices up to time t, the aim is to forecast the closing price at time t+h.

**(2) Forecasting Horizons**

Based on comparisons of the response situations of these models in different temporal contexts and under various demands for prediction, two different-length look-back periods combined with two corresponding prediction times will provide the following arrangements:

- **10-day → 1-day ahead** ($T = 10$, $h = 1$)
- **10-day → 5-day ahead** ($T = 10$, $h = 5$)
- **100-day → 1-day ahead** ($T = 100$, $h = 1$)
- **100-day → 5-day ahead** ($T = 100$, $h = 5$)

A 10-day short-term window was selected for this test; however, in the process of calculation by the system, it is usually taken as a whole week to avoid excessive noise and false alarms from single daily changes. The long input window is set to 100 trading days; that is, the entire trend of stock price changes over a longer time span will be taken into account by adjusting its trend with a slow speed adjustment for subsequent training. At the same time, for tomorrow's forecast, the extent to which the model can respond promptly to current market changes will be evaluated; As for the long-term prediction function of the model, whether it exhibits stable and adaptable characteristics over multiple time periods needs to be considered[17].

**(3) Input Window Construction**

Raw time series data cannot be directly used as training samples for supervision in supervised learning; Therefore, a sliding window approach is applied to create sequences of fixed lengths that can be fed into an algorithm[15]. Let $T$ denote the input window length. For each trading day $t$, an input window is constructed as

$$\mathbf{x}_{t-T+1:t} = (x_{t-T+1}, x_{t-T+2}, \ldots, x_t), \tag{10}$$

representing the closing prices over the past $T$ trading days. Each input window is paired with a corresponding future prediction target at horizon $h$.

**(4) Rolling Forecasting Protocol**

A rolling forecast method is used to create supervised samples for the entire dataset. At each iteration, the input window is advanced one time step ahead to generate a series of overlapping input-output pairs. This approach maintains the sequence of time and ensures that the model is tested in line with the actual application scenario for forecasted applications[28].

**3.1.3 Evaluation Metrics**

To evaluate the predictive performance of different models in stock price and index forecasting tasks, this study adopts three evaluation metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Accuracy.

**(1) Mean Absolute Error (MAE)**

The Mean Absolute Error measures the average magnitude of prediction errors without considering their direction, and is defined as[27]

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i + \hat{y}_i|, \tag{11}$$

where $yi$ and denote the ground-truth and predicted values, respectively, and $N$ is the total number of samples.

**(2) Root Mean Squared Error (RMSE)**

The Root Mean Squared Error is more sensitive to large deviations,and is defined as

$$\mathrm{RMSE} = \sqrt{\sum_{i=1}^{N}(y_i + \hat{y}_i)^2}\,. \tag{12}$$

**(3) Accuracy within** ±10%

A tolerance-based accuracy metric was developed to evaluate prediction stability. Predictions with an absolute rela-

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

tive error less than or equal to 10% of the ground truth are classified as correct:

$$\text{Accuracy} = \frac{|\{t \mid \frac{|\hat{y}_t - y_t|}{y_t}\} < 0.1|}{N} \times 100\%, \quad (13)$$

where $y_t$ and $\hat{y}_t$ denote the ground-truth and predicted values at time step $t$, respectively, and $N$ is the total number of evaluated samples.This metric captures how consistently each model produces near-accurate forecasts within a predefined relative tolerance[19].

## 3.2 Experimental Results on Predicting Nasdaq Index

This section presents a comparison of the prediction performance among MLP, LSTM, Transformer, ARIMA and SVR models for the NASDAQ Index dataset. Two look-back window lengths are considered: 100 and 10, and two forecast periods, each with a length of 1 and 5. In terms of model performance evaluation, MAE, RMSE and prediction accuracy at ±10% are adopted. Subsequently, the analysis and discussion of the results will be provided.

### 3.2.1 Overall Model Comparison

**Table 3. Forecasting Error Metrics (MAE and RMSE) on the Nasdaq Index under Different Window Lengths and Forecasting Horizons**

| Window | Horizon | Model | MAE | RMSE |
|---|---|---|---|---|
| 10 | 1 | MLP | 1058.7 | 1171.9 |
| 10 | 1 | LSTM | 267.3 | 334.5 |
| 10 | 1 | Transformer | 969.7 | 1277.4 |
| 10 | 1 | ARIMA | 202.5 | 260.2 |
| 10 | 1 | SVR | 763.0 | 1343.7 |
| 10 | 5 | MLP | 1138.9 | 1266.3 |
| 10 | 5 | LSTM | 315.5 | 396.8 |
| 10 | 5 | Transformer | 1049.9 | 1483.0 |
| 10 | 5 | ARIMA | 523.5 | 829.3 |
| 10 | 5 | SVR | 2482.2 | 3722.5 |
| 100 | 1 | MLP | 879.2 | 997.6 |
| 100 | 1 | LSTM | 293.1 | 361.7 |
| 100 | 1 | Transformer | 790.2 | 1032.8 |
| 100 | 1 | ARIMA | 157.9 | 205.2 |
| 100 | 1 | SVR | 2294.6 | 2978.1 |
| 100 | 5 | MLP | 824.5 | 979.5 |
| 100 | 5 | LSTM | 346.0 | 424.1 |
| 100 | 5 | Transformer | 986.9 | 1287.7 |
| 100 | 5 | ARIMA | 357.5 | 447.1 |
| 100 | 5 | SVR | 4220.5 | 5226.9 |

**Table 4. Forecasting Accuracy on the Nasdaq Index under Different Window Lengths and Forecasting Horizons**

| Window | Horizon | Model | Accuracy (±1%) | Accuracy (±5%) | Accuracy (±10%) |
|---|---|---|---|---|---|
| 10 | 1 | MLP | 2.2% | 21.3% | 89.3% |
| 10 | 1 | LSTM | 32.3% | 98.2% | 100.0% |
| 10 | 1 | Transformer | 8.5% | 47.4% | 82.6% |
| 10 | 1 | ARIMA | 44.9% | 97.9% | 100.0% |
| 10 | 1 | SVR | 26.6% | 69.9% | 85.4% |

| Window | Horizon | Model | Accuracy (±1%) | Accuracy (±5%) | Accuracy (±10%) |
|---|---|---|---|---|---|
| 10 | 5 | MLP | 2.1% | 19.2% | 77.6% |
| 10 | 5 | LSTM | 29.0% | 93.0% | 99.7% |
| 10 | 5 | Transformer | 10.1% | 50.2% | 79.1% |
| 10 | 5 | ARIMA | 20.5% | 76.8% | 94.7% |
| 10 | 5 | SVR | 12.8% | 40.0% | 54.8% |
| 100 | 1 | MLP | 5.8% | 35.1% | 97.9% |
| 100 | 1 | LSTM | 28.6% | 97.6% | 100.0% |
| 100 | 1 | Transformer | 10.1% | 59.7% | 83.8% |
| 100 | 1 | ARIMA | 55.9% | 99.5% | 100.0% |
| 100 | 1 | SVR | 4.2% | 18.0% | 32.6% |
| 100 | 5 | MLP | 9.0% | 47.5% | 88.4% |
| 100 | 5 | LSTM | 23.7% | 93.0% | 99.7% |
| 100 | 5 | Transformer | 9.7% | 44.0% | 80.6% |
| 100 | 5 | ARIMA | 25.2% | 88.7% | 99.6% |
| 100 | 5 | SVR | 3.8% | 12.5% | 16.7% |

As shown in Tables 3 and 4, the forecasts for each combination of window length and forecasting period obtained by these three models are presented. In terms of overall comparison for the low MAE and RMSE, it is concluded that the LSTM model performs well. However, in specific configuration situations, there might be certain deviations. Overall predictive capability is not as strong. The MLP model shows competitive performance in some scenarios, especially under the long window situation, both MAE and RMSE are significantly reduced and accuracy is improved, but it is more prone to forecast difficulty. As the prediction period extends to $h = 5$ and a small number of inputs ($T = 10$) is used for training, the performance decline of MLP becomes more pronounced. The Transformer model shows a general improvement in longer windows, but it is still less effective than LSTM; at the same time, its performance drops significantly for multi-step forecasting problems.

For the non deep learning models, the ARIMA model has rather steady baseline performance on short term forecasting task. In one-step-ahead forecasting setting ($h = 1$), ARIMA gets small MAE and RMSE, its predicting accuracy keeps being very high under all different error tolerances. But when the forecast horizon reaches $h = 5$, we can see that the error indicators for ARIMA increase significantly and there is also a small drop in prediction accuracy. The SVR model has worse overall predictability and is quite sensitive to changes in both window size and forecast horizon. In the short term forecast situation, SVR can still keep a certain amount of prediction accuracy at an error threshold with some looseness, however, it usually has a higher MAE and RMSE compared to other models. With an increase in the forecasting horizon, the prediction error for SVR rises very quickly and thus there is a considerable drop in its ability to predict accurately.

**3.2.2 Model Robustness across Forecasting Horizons**

This section compares the performance changes between next-day forecasting ($h = 1$) and five-day forecasting ($h = 5$) under fixed input window lengths ($T = 10$ or $T = 100$). Overall, as the forecasting horizon increases, the prediction task becomes more challenging for all models, which is reflected in higher error metrics and a decline in prediction accuracy.

For the deep learning models, in terms of the magnitude of performance degradation, the LSTM model demonstrates the strongest robustness. Under both window settings, its accuracy (±10%) decreases only marginally when the forecasting horizon increases from $h = 1$ to $h = 5$ (from 100.0% to 99.7% in both cases), indicating that the model is able to maintain stable temporal modeling capability over longer forecasting horizons. In contrast, the MLP model is the most sensitive to changes in forecasting horizon, with its accuracy (±10%) dropping substantially as the horizon increases, particularly under short-window

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

conditions (from 89.3% to 77.6% when $T$ = 10, and from 97.9% to 90.3% when $T$ = 100). The Transformer model lies between these two extremes: under both window length settings, its accuracy (±10%) consistently decreases as the forecasting horizon increases from $h$ = 1 to $h$ = 5, but the overall magnitude of degradation remains relatively moderate (from 82.6% to 79.1% when $T$ = 10, and from 83.8% to 80.6% when $T$ = 100).

For the two non-deep-learning models, the prediction accuracy of ARIMA also decreases when the forecasting horizon extends from $h$ = 1 to $h$ = 5, but the overall degradation is relatively limited, particularly under the longer window setting, where high stability is still preserved (when $T$ = 10, the ±10% accuracy decreases from 100.0% to 94.7%; when $T$ = 100, it decreases only slightly from 100.0% to 99.6%). This indicates that ARIMA exhibits strong stability in short-term forecasting, although it is inevitably affected by error accumulation in multi-step forecasting scenarios. In contrast, SVR is the most sensitive to changes in forecasting horizon: when the horizon increases from $h$ = 1 to $h$ = 5, its prediction accuracy declines sharply, and this phenomenon is consistently observed under different window lengths (for example, when $T$ = 10, the ±10% accuracy drops from 85.4% to 54.8%; when $T$ = 100, it drops from 32.6% to 16.7%). These results indicate that SVR exhibits weak robustness across forecasting horizons and struggles to maintain stable performance in multi-step forecasting tasks.

**3.2.3 Model Sensitivity to Window Length**

This part looks at how well the three models predict things when they use either a short window with T=10 or a long window with T=100, but they all try to figure out what will happen just one time step ahead ($h$=1) or five time steps ahead ($h$=5). Experimental results show that in all the deep learning models, MLP is most sensitive to input window length change: when we increase the input window from 10 trading days to 100 trading days, we see both MAE and RMSE drop considerably as well as noticeable improvement on predictability; such a boost is more pronounced for multi step forecasting (i.e., large decrease of MAE and RMSE & significant rise in accuracy). On the other hand, it is also observed that the transformer model gains advantages when given longer inputs as well, where its errors are usually lower for $T$=100 than $T$=10, but the magnitude of improvement in the multi step setting is still fairly small. On the contrary, for the LSTM model it has less sensitivity towards change of window length: even when set on shorter windows, its errors are still very low; additionally, increasing the window size actually causes only minor growths in MAE/RMSE (e.g., with $h$=5, MAE grows from 315.5 up to 346.0) but predictions stay accurate all along. Which means that the LSTM can still be good at picking up those discriminative temporal dependence patterns from a rather short history sequence.

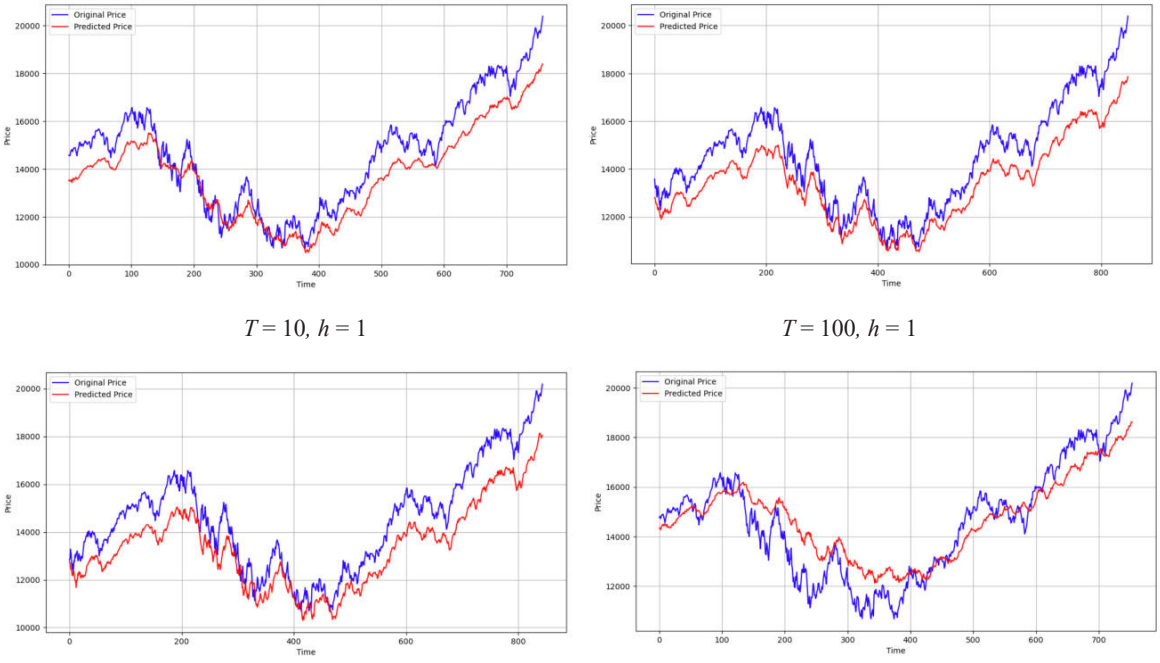
For the non-deep-learning models, ARIMA also exhibits sensitivity to changes in input window length, but its trend differs from that of deep learning models. When the input window is extended from $T$ = 10 to $T$ = 100, ARIMA shows an overall decrease in MAE and RMSE under both forecasting horizon settings, indicating that longer historical information helps capture linear temporal dependencies (for example, when $h$ = 1, MAE decreases from 202.5 to 157.9, and RMSE decreases from 260.2 to 205.2). Prediction accuracy also improves with increasing window length, particularly in the next-day forecasting scenario, where the ±10% accuracy remains close to 100% under different window settings. However, for the multi-step forecasting task ($h$=5), even though a larger window still results in an increase in the level of accuracy (from 94.7% to 99.6%), the error metrics are still substantially greater than those from single step forecasting, which implies that ARIMA will still inevitably be affected by the accumulation of errors during multi-step forecast tasks. On the contrary, SVR has very strong sensitivity to changes in the length of the input window length and its prediction results will be much worse when the length is longer. As for the case where T goes from 10 to 100, both MAE and RMSE grow greatly in both forecast horizon circumstances, which means that adding more historical sequence information doesn't enhance the model's performance; rather it makes its predictive power worse (e.g., if h equals 1 then MAE jumps up from 763. 0 to 2294. 6 while RMSE rises from 1343. 7 all the way to 2978. 1). We can also see that this trend is reflected in the prediction accuracy, particularly for multi-step ahead predictions; as shown below, when the length of the window increases, the reduction in prediction accuracy is relatively pronounced. At this point, it has become increasingly obvious. For example, at a value of 5 for h, it is reduced from 54.8% to 16.7%. It is found that SVR has not used the long-term historical data

well; It is also highly responsive to variations in the window Size.

**3.2.4 Visualization of experimental results**

Under the combined effect of different T values (10, 100) and h values (1, 5), the corresponding fitted curves for each model are displayed. There are two curves in each Figure: the blue curve represents the actual NASDAQ Index values; and the Red Curve is the predicted 15values by the model. By observing visually, that is, by checking these plots; whether each model could identify the overall trend of the time series; local fluctuations and other characteristics within the data. Because these metrics do not fully capture the entire dynamic range of changes as shown in this type of graph.

**MLP**




$T = 10,\ h = 1$ $T = 100,\ h = 1$

$T = 10,\ h = 5$ $T = 100,\ h = 5$

**Figure 6. Fitted curves of the MLP model under different window lengths ($T$ = 10, 100) and forecasting horizons ($h$ = 1, 5) on the Nasdaq index.**

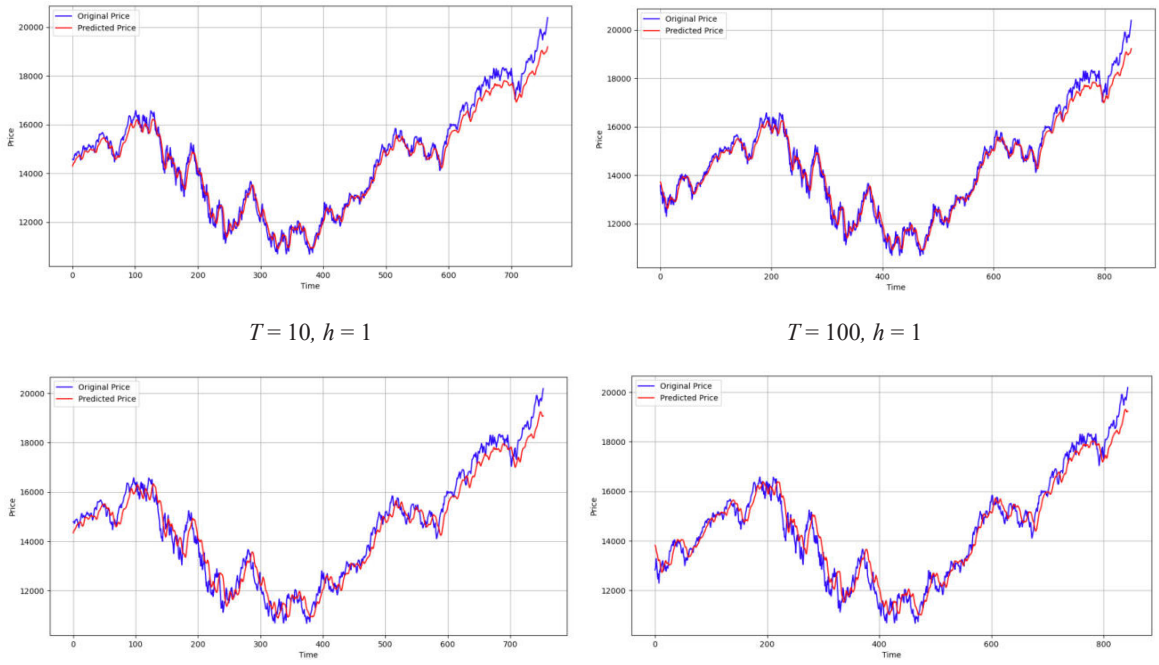
From Figure 6 we can see that the MLP model is able to capture the general trend of the Nasdaq index for different window sizes and forecast horizons but its prediction curve tends to be much smoother than the actual truth. In terms of the next-day forecast setting ($h$=1), the MLP model does follow along with the long term trend alright, however it also seems to be under predicting some of the local bumps and turns. At an extended forecast horizon of $h$=5, there is a larger degree of smoothing and temporal lag present in the predicted values, as well as short-term oscillations that appear to be more compressed, especially when using the short window size $T$=10. On the other hand, when using $T$=100(long window), it partially solves this problem which means that the MLP model relies very much on how many years of history input we provide as it does not perform well in more difficult forecasting situations.

As shown in Figure 7, we can see that the LSTM model was able to follow along with the general movement of the Nasdaq index on all 4 different experiments. Under the next-day forecasting setting $h$=1, it can be observed that the predicted series is very close to the ground truth curve, which means that the model is able to grasp the overall trend correctly as well as the local fluctuation and turning point. When $h$ = 5, the forecasted curve shows some smoothing and small lag in certain regions where there is a quick fluctuation but it is still very close to the true index movement. Also, different lengths of input windows do not greatly affect the fitted pattern of an LSTM model; hence we can say that it is still able to pick up useful tem-

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

poral information even when given only a short amount of historical context.

**LSTM**

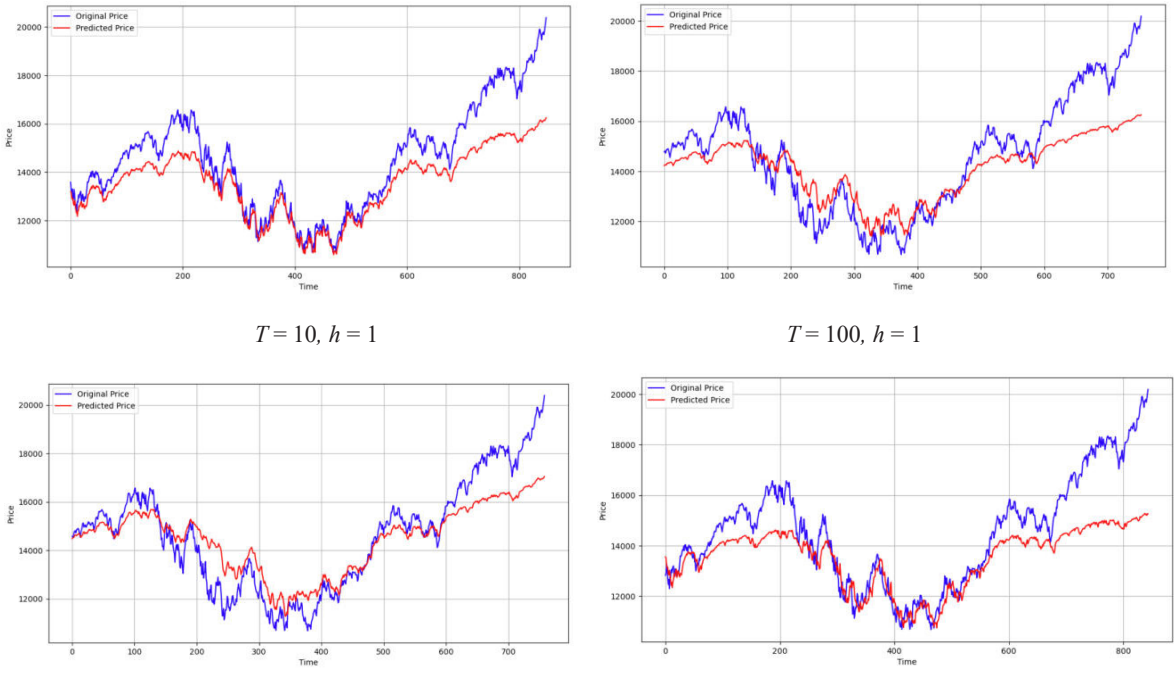



$T = 10,\ h = 1$ $T = 100,\ h = 1$

$T = 10,\ h = 5$ $T = 100,\ h = 5$

**Figure 7. Fitted curves of the LSTM model under different window lengths ($T$ = 10, 100) and forecasting horizons ($h$ = 1, 5) on the Nasdaq index.**

**Transformer**




$T = 10,\ h = 1$ $T = 100,\ h = 1$

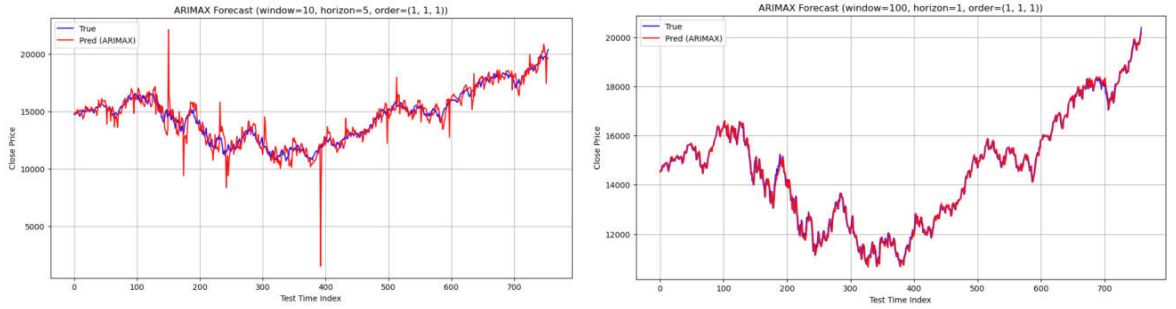
$T = 10,\ h = 5$ $T = 100,\ h = 5$

**Figure 8. Fitted curves of the Transformer model under different window lengths ($T$ = 10, 100)**

**and forecasting horizons ($h$ = 1, 5) on the Nasdaq index.**

From Figure 8 we can see that the Transformer model could also learn about the general movement of the Nasdaq index under various window lengths and forecast horizons but it tends to have a smoother predicted curve. Under the next day forecasting setup where $h$=1, Transformer is fairly aligned with the long term trend but shows under prediction on local fluctuations as well as extreme points. As for when the forecast horizon is extended to $h$=5, its smoothing effect gets even stronger and some short-term changes cannot be well reflected. On the other hand, for $T$=100 which is the long window setting, there is some improvement in terms of trend fitting, but the model is still not able to fit high frequency fluctuation well. From these we can infer that the transformer does better job of representing global trends on this problem but has somewhat weaker ability to represent short term fine grained dynamics.

**ARIMA**




$T$ = 10, $h$ = 1 $T$ = 100, $h$ = 1

$T$ = 10, $h$ = 5 $T$ = 100, $h$ = 5

**Figure 9. Fitted curves of the ARIMA model under different window lengths ($T$ = 10, 100) and forecasting horizons ($h$ = 1, 5) on the Nasdaq index.**

From Figure 9, the ARIMA model can closely follow the actual changes of the Nasdaq Index at various input window lengths and forecast periods, especially in the next-day prediction scenario ($h$=1), it shows relatively stable performance. For both the short window ($T$ = 10) and the long window ($T$ = 100), the predicted curve of ARIMA is very similar to the actual price series, indicating that it has a relatively strong ability for linear-time dependency model building and short-term dynamic prediction. At this point when we set the forecasting horizon as $h$=5, one can clearly see that the predictive stability of the ARIMA model has decreased quite a bit, and more notably within the short window scenario, there's an evident fluctuation and deviation present on the predicted curve, which demonstrates the accumulation error from multi step ahead forecast. $T$=100 when the long window is configured, it has only partly improved the general trend fitting and made the predicted curves more similar to the actual direction of movement but still fails to eliminate prediction instability. In general, ARIMA is better suited

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

for short term forecasting problems, but it's not so good in terms of predictive performance and robustness for multistep forecasting, especially when the window size is small.

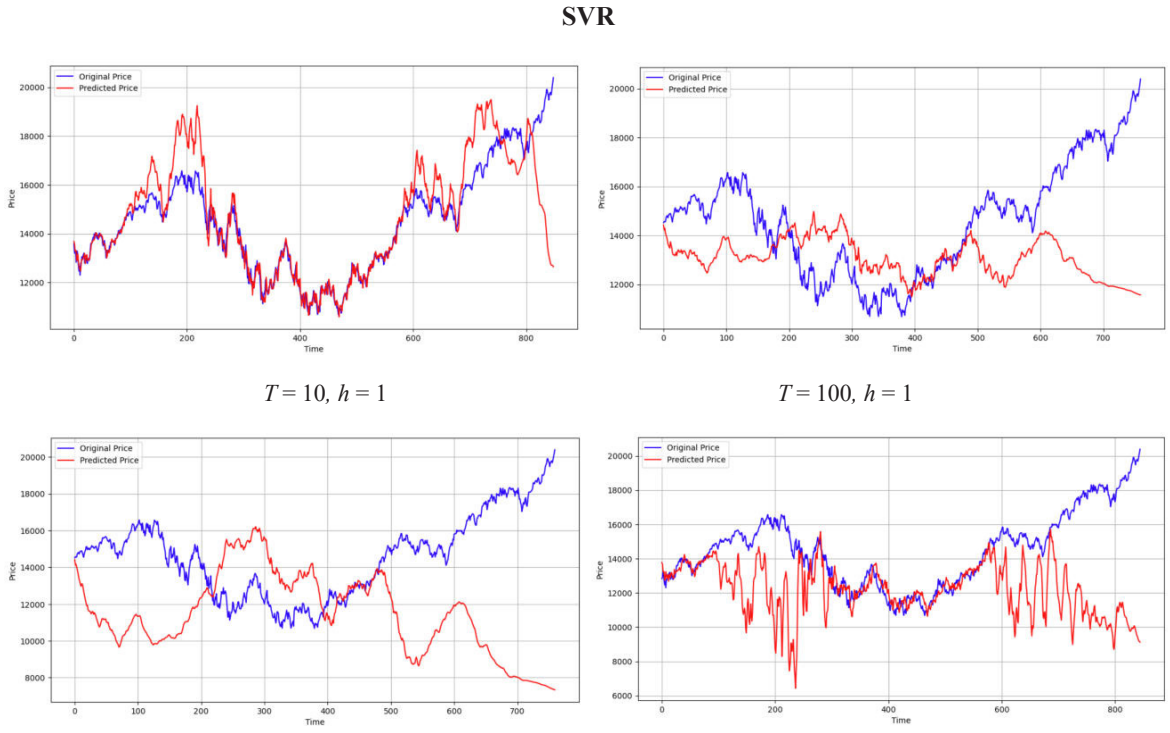



**Figure 10. Fitted curves of the SVR model under different window lengths ($T$ = 10, 100) and forecasting horizons ($h$ = 1, 5) on the Nasdaq index.**

From Figure 10, it can be seen from the above figure that, at most times, there are significant deviations between the predicted curve of the SVR model and that of the actual Nasdaq Index; in other words, the SVR model has weak fitting capability for complex patterns or long-term trends within the selected time range. Under any circumstances, such as a small difference at both ends or some outliers on the line chart, these features remain consistent with other cases where the data trend may show deviations after being fed into a specific model. When the next-day forecast mode is enabled (h=1), while it can generally conform to the trend of the overall data through a small number of observations in this window size (T=10). However, its predicted curve shows significant fluctuations and prominent local peaks. Under the long-window setting (T=100), there are significant deviations from reality in terms of both trends and fluctuations; at this point, it is difficult for the model to capture the overall trend. As the prediction time h increases, the forecast error will increase rapidly. When it comes to a short window, we can see considerable volatility and instability in the predicted curves; they cannot truly reflect the actual price changes; With respect to a long Window, there is a large discrepancy between its predictions for the entire range of data and the actual series; it failed to catch the general trend of the primary change pattern of, therefore, had obvious prediction distortion. In summary, SVR has a weakly stable and robust performance under various window-length combinations and multi-step forecasts; thus, it cannot well capture the dynamic development characteristics of complex time series and is not appropriate for performing multiple-step-ahead predictions.

### 3.3 Experimental Results on Predicting Individual Stocks

This part compares the prediction performance of the MLP, LSTM, Transformer, ARIMA and SVR models on each of the six stocks. All the experiments use a lookback window of 100 trading days and forecast for one day ahead, that is, each experiment has a training period of 100 days, and then predicts the next day's performance.

The performance of the model is assessed by mean absolute error (MAE), root mean squared error (RMSE), and so on, and its prediction accuracy has reached a ±10% range. Subsequently, it is analysed and examined the results.

**3.3.1 Model Comparison**

**Table 5. Stock-level forecasting performance under the 100-day input window and 1-day forecasting horizon.**

| Stock | Metric | MLP | LSTM | Transformer | ARIMA | SVR |
|---|---|---|---|---|---|---|
| Amazon | MAE | 12.1 | 3.2 | 7.1 | 2.5 | 21.2 |
| | RMSE | 13.3 | 4.2 | 8.4 | 3.4 | 34.6 |
| | Accuracy (±1%) | 1.6% | 34.2% | 9.3% | 42.0% | 11.1% |
| | Accuracy (±5%) | 22.6% | 91.9% | 58.2% | 97.0% | 45.7% |
| | Accuracy (±10%) | 78.5% | 99.0% | 97.4% | 99.6% | 68.1% |
| Apple | MAE | 26.9 | 7.8 | 12.5 | 2.3 | 54.1 |
| | RMSE | 29.6 | 9.3 | 16.9 | 3.4 | 71.8 |
| | Accuracy (±1%) | 1.2% | 9.4% | 11.4% | 56.4% | 4.2% |
| | Accuracy (±5%) | 5.4% | 72.7% | 59.6% | 98.3% | 23.2% |
| | Accuracy (±10%) | 22.0% | 99.6% | 74.4% | 99.7% | 36.0% |
| Meta | MAE | 41.5 | 27.0 | 83.0 | 7.0 | 202.8 |
| | RMSE | 52.3 | 37.1 | 122.0 | 10.8 | 277.3 |
| | Accuracy (±1%) | 4.0% | 10.0% | 9.6% | 43.1% | 5.4% |
| | Accuracy (±5%) | 24.4% | 53.6% | 38.1% | 96.2% | 25.8% |
| | Accuracy (±10%) | 65.1% | 90.0% | 45.2% | 99.1% | 38.0% |
| Microsoft | MAE | 15.7 | 10.6 | 31.3 | 4.1 | 169.5 |
| | RMSE | 19.0 | 13.2 | 44.1 | 5.6 | 219.3 |
| | Accuracy (±1%) | 13.3% | 19.0% | 13.4% | 56.2% | 5.3% |
| | Accuracy (±5%) | 62.1% | 91.3% | 44.6% | 98.8% | 26.8% |
| | Accuracy (±10%) | 89.3% | 100.0% | 75.5% | 100.0% | 35.5% |
| NVIDIA | MAE | 7.0 | 17.2 | 41.8 | 2.0 | 69.7 |
| | RMSE | 8.9 | 24.5 | 56.0 | 3.1 | 88.0 |
| | Accuracy (±1%) | 2.8% | 4.9% | 3.0% | 30.6% | 2.9% |
| | Accuracy (±5%) | 22.6% | 26.7% | 14.5% | 90.5% | 10.9% |
| | Accuracy (±10%) | 57.6% | 44.4% | 20.3% | 98.7% | 17.8% |
| Tesla | MAE | 30.5 | 8.6 | 14.0 | 7.3 | 30.7 |
| | RMSE | 41.2 | 11.9 | 17.4 | 10.1 | 41.8 |
| | Accuracy (±1%) | 5.6% | 19.1% | 8.5% | 23.5% | 7.0% |
| | Accuracy (±5%) | 27.1% | 76.0% | 49.6% | 82.4% | 31.7% |
| | Accuracy (±10%) | 53.2% | 95.8% | 84.0% | 97.1% | 54.4% |

Table 5 presents the stock-level forecasting performance of the five models under the same experimental setting, where all models employ a 100-day input window and a one-day forecasting horizon. As shown in the table, the LSTM model achieves lower MAE and RMSE on most individual stocks and attains the best overall performance on the accuracy (±10%) metric. For instance, on stocks such as Amazon, Apple, Microsoft, and Tesla, the prediction accuracy of LSTM is generally close to 100%, with error magnitudes consistently smaller than those of the other two models. This means that LSTM has strong prediction ability and good consistency on different stocks.

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

On the contrary, the performance of the MLP model is very unstable. But it still has acceptable errors and predictions for some stocks like Amazon and Microsoft, but when we look at others such as Apple, it's only accurate 22.0%, so it seems very sensitive to the actual underlying characteristics of different stock prices. The transformer model is also exhibiting unstable behavior: it gets fairly high accuracy for Amazon and Tesla, but the error on prediction grows quite a lot and the accuracy drops sharply when more volatile stocks like Meta or NVIDIA are used.

Non-deep learning models, ARIMA shows more stable and consistent predictive performance on a per-stock basis. ARIMA gets very small MAE and RMSE for most stocks when the input is 100-day and the forecast is for 1 day, its prediction result remains fairly accurate-especially within +- 10% of tolerance, it's still quite high, there are also some stocks close to or even up to 100%. Therefore, it can be seen that given enough history ARIMA was able to pick up on the short term pricing dynamics and also performs fairly consistently across different equities. On the other hand, SVR shows much poorer overall performance at the stock level and has considerable variation for different stocks. Most of its MAE and RMSE are significantly larger than those of other models, and the degree to which prediction accuracy changes with stock-specific factors differs greatly. SVR has a strong response to structural changes and volatility in an individual stock's price series; therefore, it cannot provide stable and reliable predictions of the company at this time.

NVIDIA stands out clearly, as all other models perform poorly for it when applied using ARIMA. It can be seen that some individuals of the prices are significantly affected by external factors, while there are problems with keeping up with such frequent changes in forecast models. Therefore, this study also found that there is considerable heterogeneity in the form of stocks' forecasts, and findings based on these results cannot be directly applicable to single companies.

### 3.3.2 Visualiz ation of experimental results

**Amazon**

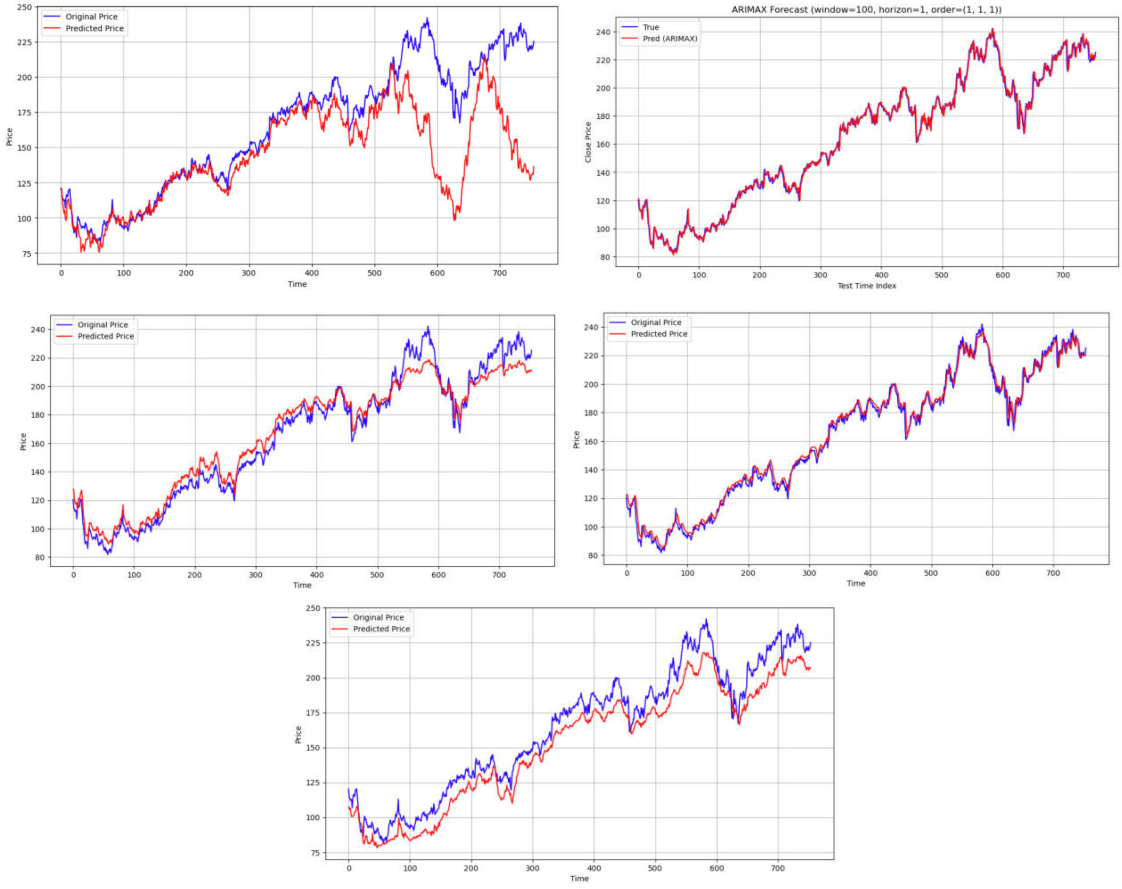


**Figure 11. Fitted curves of five models on Amazon stock under a 100-day input window and a one-day forecasting horizon.**

**Apple**

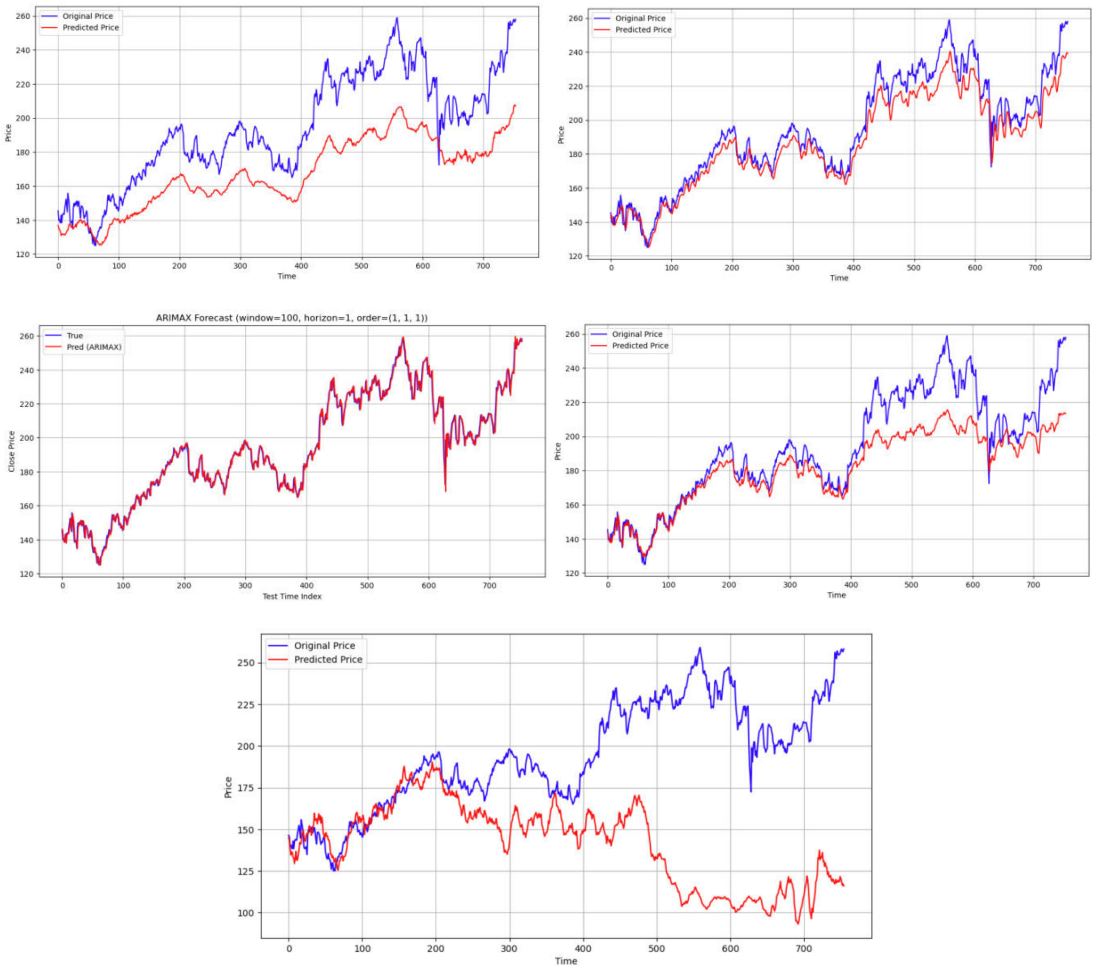
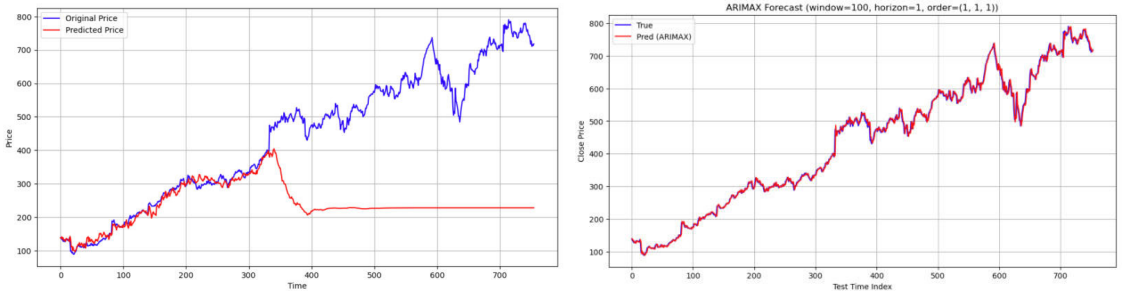



**Figure 12. Fitted curves of five models on Apple stock under a 100-day input window and a one-day forecasting horizon.**

**Meta**



ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

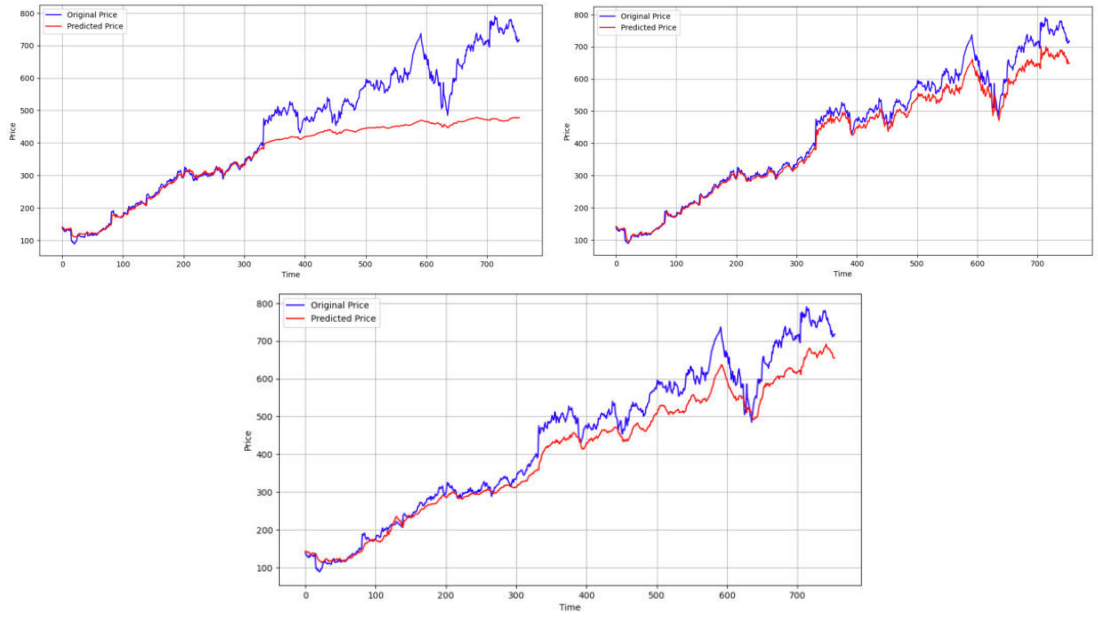



**Figure 13. Fitted curves of five models on Meta stock under a 100-day input window and a one-day forecasting horizon.**

**Microsoft**

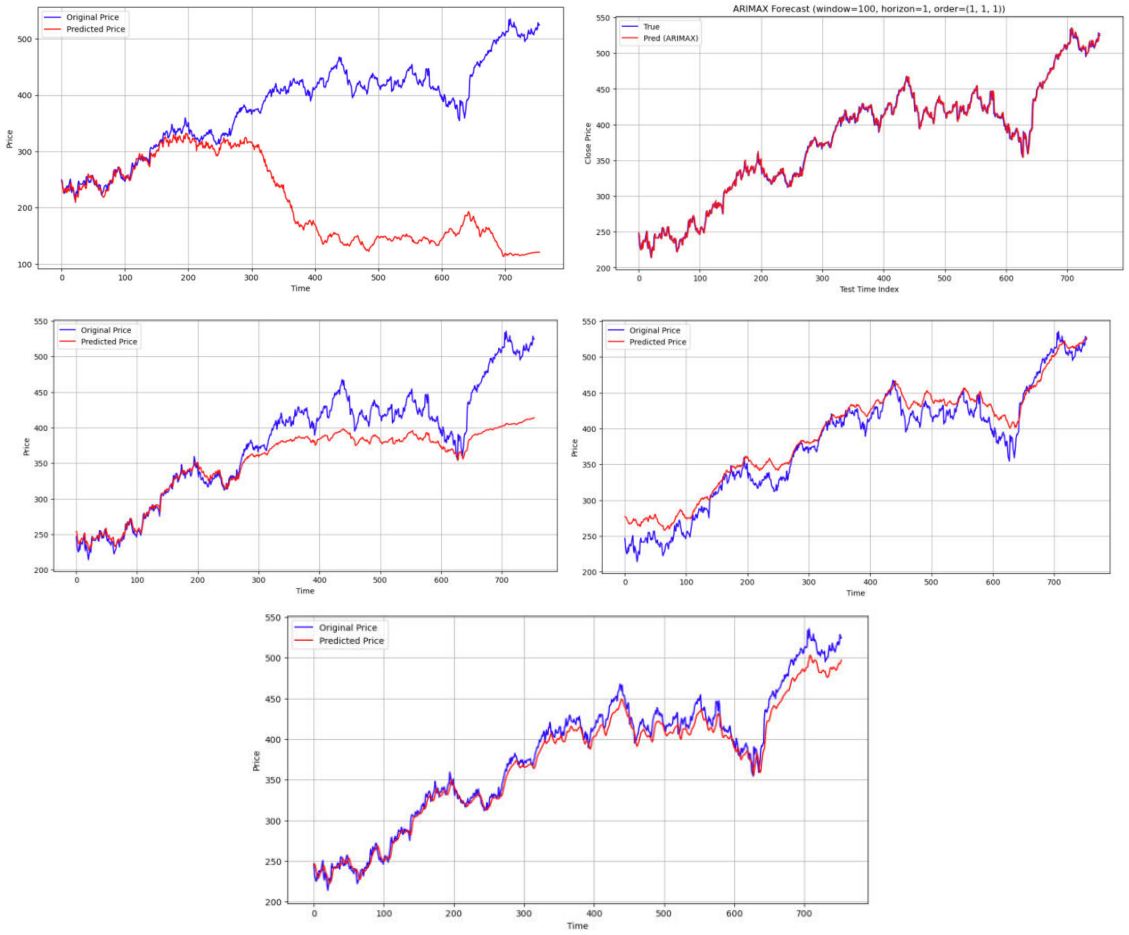
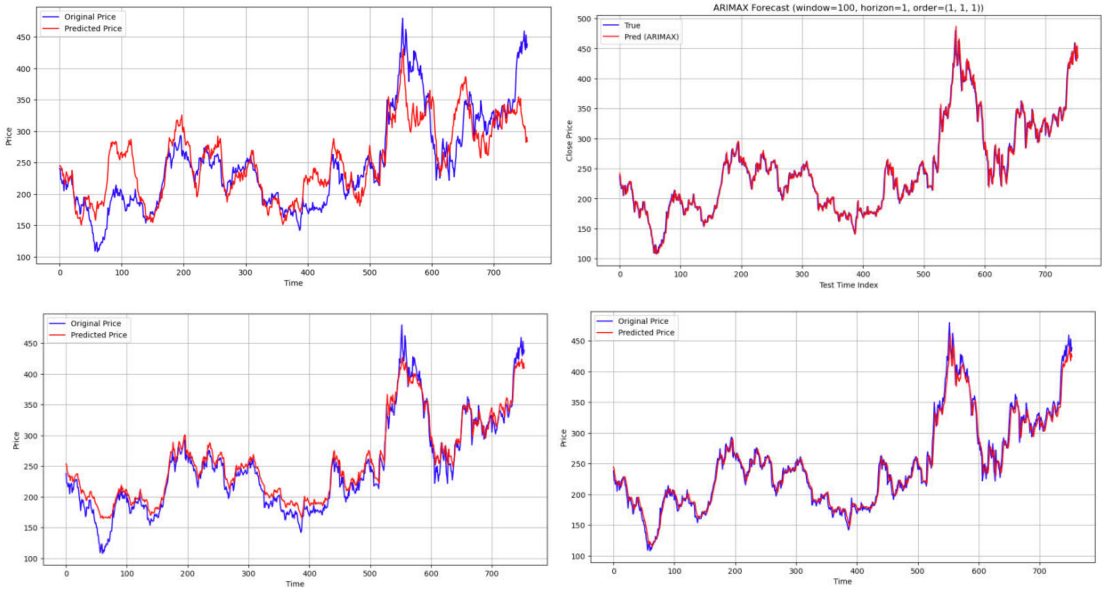



**Figure 14. Fitted curves of five models on Microsoft stock under a 100-day input window and a one-day forecasting horizon.**

**NVIDIA**


**Figure 15. Fitted curves of five models on NVIDIA stock under a 100-day input window and a one-day forecasting horizon.**

**Tesla**



ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

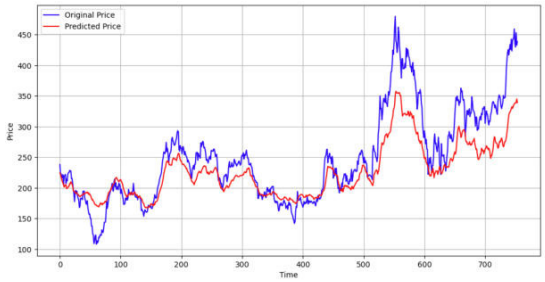



**Figure 16. Fitted curves of five models on Tesla stock under a 100-day input window and a one-day forecasting horizon.**

The six figures above (Figure 11, Figure 12, Figure 13, Figure 14, Figure 15 and Figure 16) provide a direct visual comparison of the predictive fitting performance of the five models across different individual stocks. Overall, although all models are able to capture the general movement of stock price series to some extent, their fitting quality and stability vary substantially across both models and stocks.

As can be observed from the figures, the five models share several common characteristics in stock-level forecasting. Under the condition that there is no large-scale fluctuation and a generalised trajectory of price changes can be obtained, all three kinds of models' predictions for the price trajectories generally align closely with its actual movement trend from the beginning until the end. Based on the above analysis, it can be known that both deep-learning algorithms and traditional statistical models / Methods (such as ARIMA) have an ability to extract meaningful trend information in a limited number of historical data for short-term predictions. However, when the price series have large fluctuations, structural changes, or high-frequency volatility, the performance difference between different models will be more pronounced at this time. Among all of the models for these four stocks, LSTM exhibits the best stability. The predicted curve is very close to the actual price; its tendency over a long period is stable; the deviation in terms of stability at high frequency occurs less frequently. And it can be observed from stocks such as Amazon, Apple, Microsoft and Tesla, as well as a higher accuracy rate for errors of LSTM in Table 5; compared with the transformation model's tendency to be smooth, it generally underestimates the extent of changes in prices due to sudden jumps and produces local deviations. However, there are problems with the MLP model; that is, it cannot adapt well to changes in some stocks and has a strong fluctuation tendency when faced with other types of companies. Therefore, from this aspect, its stability is lower than that of other models.

Traditional models also exhibit significantly distinct behaviours when displayed in visualisations. For most of the stocks, ARIMA generates predicted curves that fit well with the actual prices; however, in cases where there is a clear trend present, such as some high-frequency data series, it may have limitations and fail to capture these trends accurately. However, SVR produced predictions that were noticeably inconsistent with the actual price series of several stocks and exhibited an extremely high degree of volatility or persistently incorrect trends. NVIDIA's high-volatility companies, which have significant problems with their errors and low precision obtained from a quantitative analysis, respectively occur frequently in that data set.

Different features of individual stocks also contribute to the detected fluctuations in the performance of the predicted models. Nvidia is one of the most typical cases; other than using ARIMA, none of these models could adequately capture price fluctuations that rapidly appeared after introducing new technologies, highlighting limitations during the adjustment period of high-frequency noise or structural changes. Compared with equities such as Amazon, Apple and Microsoft that have more stable prices and price fluctuations; some algorithm results will be somewhat inconsistent.

Finally, considering the graphically presented equity-level projections also verifies the results of the calculations in terms of numbers. The performance and stability benefits of some frameworks at the benchmark level may not be reflected in individual equity securities. Due to different features of patterns recognition, fluctuation control and integrity protection across several listed companies. As

shown in the above results, it can also be used for both graph drawing at the time and later qualitative judgement through quantitation; moreover, it has also been introduced into many prediction-target economic scenarios.

### 3.4 Comparison between Index-level and Stock-level Forecasting

This section is responsible for comparing and analysing the forecasts at the index and stock levels to reveal their similar and distinct features patterns displayed by different models in both prediction tasks, as well as providing suggestions for generalisation and application based on this.

#### 3.4.1 Shared Patterns across Index and Stock Forecasting

Evaluate the index-level forecast experiment result (Table 3 and Table 4) with the stock-level forecast experiment result (Table 5), aiming to summarise which models have similar character under the prediction task. Although index prediction and stock prediction have different data structures and volatility features, some of the model's relative performances are still maintained between them.

**(1) LSTM:** LSTM is more stable and consistent at both the index level and the stock level for prediction. In terms of the index level, LSTM has achieved a high degree of robustness in accuracy (approximately 100 percent) at different window lengths and forecasting horizons, with MAE and RMSE generally below low values. When it comes to single-stock Data, LSTMs perform well or almost as well as other methods across most individual stocks, and there is relatively little difference in performance when applied to different equities. Therefore, it can be concluded that the LSTM has achieved good results in handling smoothed index series. At the same time, its temporal modelling ability is also suitable for single-stock short-term prediction problems.

**(2) MLP:** The MLP model has shown significant instability and sensitivity in both index-level and stock-level predictions; at the index level, its prediction performance is highly responsive to variations in input window length and forecasting horizons. In terms of individual stocks, their performance varies significantly at the stock level; while acceptable performance has been achieved for some stocks (such as Amazon and Microsoft), the prediction accuracy has dropped sharply for others (such as NVIDIA). The shared phenomenon suggests that MLP has a high degree of dependence on the structural characteristics of time series and is prone to significant fluctuations in predictive performance across various data patterns.

**(3) Transformer:** The Transformer model exhibits the common trait of having a strong trend-fitting ability but poor stability in both forecast tasks. The transformer captures the overall market trend of an index relatively well; however, it has difficulty performing local variation model fitting or carrying out multi-step forecasts. In stock level prediction, it has shown a certain degree of variability in different stocks Volatility; some Equities have achieved relatively favourable results, but the error for highly volatile Equities has been much higher. Although the transformer can capture general laws of change in time; but in this case, due to its limitation on representing details of changes among many types of prediction targets simultaneously.

**(4) ARIMA:** The ARIMA model has exhibited more stable short-term forecasting performance at the level of both the index and stocks. ARIMA is a classic statistical model at the index level that has achieved low forecast errors and high accuracy for short-term forecasting tasks. In terms of individual stocks at the stock level, it also has a relatively low MAE and RMSE under most circumstances, maintaining consistent stability in accuracy. The shared pattern suggests that ARIMA's ability to model linear temporal dependencies is applicable to both index and stock series, especially for short-term forecasting.

**(5) SVR:** The SVR model shows weak and unstable performance at both the index level and the stock level in forecast results. In terms of index levels, SVR has much larger prediction errors than other models and is very sensitive to changes in the forecasting horizon. Stocks at the stock level perform poorly, with high errors and low accuracy on most equities. Given that this type of behaviour suggests there may be an insufficient ability to capture the non-stationarity and dynamic changes of financial time series within the static regression framework of SVR, leading to weak generalisation capacity at different forecast levels.

#### 3.4.2 Key Differences and Limitations

Although some model features have shown consistency at both the index level and the stock level in the previous section, there are still several key differences between these two types of prediction problems; they indicate that the applicability boundaries and potential limitations of

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

these models' prediction capabilities need to be clarified.

First, index-level forecasting can reduce the volatility of data and complexity of structure to some extent, thereby decreasing the overall difficulty of this prediction problem. As the aggregation of several individual stock prices, index series have relatively smooth fluctuations and are more conducive to capturing general trends. On the other hand, there are more non-stationary and heterogeneous characteristics in a single stock price series, so its performance varies significantly from one stock to another. The phenomenon of particularity in stocks, such as NVIDIA, is observed; therefore, robust predictive performance at the index level may not guarantee stable application of a model to the stock-level forecast task.

Second, the relative advantages of various models change when moving from an index level to a stock level in prediction. For instance, ARIMA is generally used as a robust benchmark model for index prediction. However, its short-term predictive ability can be comparable to or even exceed that of deep learning models when applied to single stock forecasts. This difference indicates that the statistical model with linear time dependency is still very effective for short-term stock-level prediction problems; however, its weakness compared to deep-learning-based approaches in index prediction does not necessarily apply to individual stocks.

Additionally, some models have a higher degree of model failure at the stock level. The static regression model, such as SVR, has obvious deficiencies in index forecast. In the task of stock-level prediction, this deficiency is further exacerbated, and the error increases significantly, and the predicted curve deviates greatly from the actual price trajectory. It has a high sensitivity to structural changes and fluctuations of individual stock prices; thus, any deficiencies at the index level will be exaggerated at the stock level.

At the stage of index and stock tasks, due to some differences in difficulty levels, formats for presenting data and models applied, etc., there is a large gap.

**3.4.3 Implications for Model Generalization**

There are common patterns and considerable differences in the levels of indices and stocks' predictions; therefore, these predictive models can be used as a reference for other problems in actual applications of financial forecasting. First, the forecast results at the index level do not equate to the model's performance on individual stock forecasts. Although some models have shown good stability and performance in index prediction, their performance on stock-level tasks is generally unstable due to differences between individual stocks. This means that when using model selection based on index-level forecast results in practice, it is easy to under-estimate the uncertainty and prediction risk at the stock level. Second, the relative strengths of various models are not constant across different tasks of index and stock forecasts but rather determined by the hierarchical features of the forecast targets. Although deep learning models have demonstrated advantages over other types of neural networks in index prediction, statistical models based on linear time dependence may also perform comparably well in the short-term stock-level forecasting task. Based on these results, it can be concluded that the performance of the model needs to be judged together with the specific object of prediction, the length of time and the characteristics of volatility, rather than believing that one model always performs better than others in all forecasting tasks. In addition, the results show that stable performance at the aggregated level may mask failure risks at the individual level. Since the index series consists of numerous stocks, its forecast results are often at an "averagistic" level; On the other hand, the stock level is more likely to reveal that the model has deficiencies under high volatility and structural changes. Therefore, when conducting actual financial forecast work, we should also consider whether the model can be applied at both the index and stock levels; that is, whether there are some unreliability or instability problems in applying this kind of model to real investment analysis.

In summary, there are both differences and similarities in index- and stock-level prediction tasks at the level of difficulty and adaptability. At the same time, this research provides empirical evidence to guide the rational application of these prediction results.

## 4 Conclusion

In this paper we attempt to fill a gap in the literature around financial time series forecasting: unified benchmarks for both index-level and stock-level prediction tasks have been somewhat lacking. In light of which, within a uniform set-up for experiments, the present work

has conducted systematic evaluations of the performance of several representative predictive models at both the level and stock levels of predictions in order to obtain more comparative and understandable empirical results.

From the perspective of experimental design, we build up an unified evaluation pipeline for two kinds of prediction situations. It contained the Nasdaq index time series along with 6 typical individual stocks(Amazon,Apple,Meta,Microsoft,NVIDIA,Tesla). Regarding the choice of model, 3 popular deep learning models (MLP, LSTM, Transformer), which were then benchmarked against two other baselines that are more traditional (ARIMA and SVR). Rolling window forecasting was used so that the forecast is closer to how it would happen in real life for financial forecasting. Model performance is judged based on MAE, RMSE as well as the prediction accuracy at various relative error tolerance levels so that it can be compared fairly between different methods using the same set of evaluation standards.

The experiments also show that the difference in performance between different forecasting models on financial time-series prediction mainly depends on its structure; and the empirical conclusion drawn from indexes cannot be applied to stocks. In the index forecasting task, among deep learning model, LSTM shows the best all around stability no matter what prediction setting we use. On the contrary, the Transformer models generated rather smooth predictions, which meant that they were good at capturing long term trends but had difficulty in modeling local variation when there was very high volatility. MLP's performance was quite sensitive to changes in the input window length and forecasting horizon. This indicates that it is very responsive to different prediction setups. As for the non-deep learning model, ARIMA is very stable in short term index forecasting task and its prediction error remains quite low under most circumstances. It shows the structural benefit of a statistical model with difference and stationarity assumptions for capturing relatively smooth index level dynamics. On the other hand, SVR showed considerable performance drops with more than a certain amount of historical input and thus displayed structural limitations for static kernel based regression methods to model temporal dependencies within high dimensional feature space. Stocks, at the level of individual stock price series, which was already highly heterogeneous, increased model performance disparity even more. There were substantial differences in terms of volatility, trends and noise among different stocks. Thus it is not guaranteed that a model which performs well on an index-level forecast will do so as well when predicting at the individual stock level. And this result indicated that when applying it, we need to choose models according to the features of certain prediction objects.

In general, it is shown that the performance of forecasting models heavily relies on the hierarchical structure of the prediction objective, thus empirical conclusions based on index-level forecasting can't be easily generalized for stock-level applications. Therefore, in terms of practical application in finance for forecasting and model deployment, we need to customize our choices between models depending on the prediction target's level of volatility, its characteristic volatility over time, as well as how far into the future it needs to forecast-not simply pick the one that looked best on average from a single data set.

## References

[1] William F. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance*, 19(3):425–442, 1964.

[2] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[3] Rob J Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2021.

[4] John Y. Campbell, Martin Lettau, Burton G. Malkiel, and Yexiao Xu. Have individual stocks become more volatile? an empirical exploration of idiosyncratic risk. *The Journal of Finance*, 56(1):1–43, 2001.

[5] Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1(2):223–236, 2001.

[6] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1970.

[7] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[8] Alex J Smola and Bernhard Sch¨olkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[9] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[10] Ajith Abraham and Danil Prokhorov. Applying mlp neural network for forecasting nasdaq index. In *International Conference on Computational Intelligence for Financial Engineering*, pages 1–6.IEEE, 2002.

[11] Akanksha K. Lamba, Prateek Sharma, and Ravi Kumar.

ISSN 3106-5783 (Print) ISSN 3106-7042 (Online)

The nasdaq composite index prediction using lstm and bi-lstm multivariate deep learning approaches. In *Intelligent Systems Design and Applications, pages* 78–85. Springer, 2024.

[12] Sepp Hochreiter and J¨urgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[13] Eyas Gaffar Ahmed Osman and Faisal A. Otaibi. Integrating deep learning and econometrics for stock price prediction: A comprehensive comparison of lstm, transformers, and traditional time series models. *Machine Learning with Applications*, 22:100730, 2025.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[15] Christoph Bergmeir and Jos´e M Ben´ıtez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012.

[16] Andrew Ang, Robert J Hodrick, Yuhang Xing, and Xiaoyan Zhang. The cross-section of volatility and expected returns. *Journal of Finance*, 61(1):259–299, 2006.

[17] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. PLoS ONE, 13(3):e0194889, 2018.

[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are uni-versal approximators. *Neural Networks*, 2(5):359–366, 1989.

[19] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[20] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wenkai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *AAAI*, 35(12):11106–11115, 2021.

[21] Bryan Lim, Sercan O Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[22] Ailing Zeng, Ming Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9):11121–11129, 2023.

[23] James D Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

[24] Ruey S Tsay. *Analysis of Financial Time Series*. Wiley, 2010.

[25] Bernhard Sch¨olkopf and Alex J Smola. *Learning with Kernels*. MIT Press, 2002.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[27] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[28] Leonard J Tashman. Out-of-sample tests of forecasting accuracy: an analysis and review. *International Journal of Forecasting*, 16(4):437–450, 2000.

[29] NASDAQ. Nasdaq composite index historical data, 2024. Accessed: July 2024.