Article

Online Hybrid Neural Network for Stock Prices Prediction: A Case Study of High-frequency Stock Trading in China Market

Chengyu Li , Luyi Shen , Guoqi Qian *

Posted Date: 1 February 2023

doi: 10.20944/preprints202302.0004.v1

Keywords: High-frequency Limit Order Book; Online Fast Prediction; Hybrid Neural Network

Article

Online Hybrid Neural Network for Stock Price prediction: A Case Study of High-frequency Stock Trading in China Market

Chengyu Li , Luyi Shen and Guoqi Qian *

School of Mathematics and Statistics, The University of Melbourne, Parkville VIC 3010, Australia; chengyu.li@unimelb.edu.au; luyshen@student.unimelb.edu.au; qguoqi@unimelb.edu.au

* Correspondence: qguoqi@unimelb.edu.au (G. Qian)

Abstract: Time series data having low signal-to-noise ratio, non-stationarity and non-linearity are commonly seen in high-frequency stock trading, where the objective is to increase the likelihood of profit by taking advantage of tiny discrepancies in prices and trading on them quickly and in huge quantities. For this purpose, it is essential to apply a trading method that is capable of fast and accurate prediction from such time series data. In this paper, we develop an online time series forecasting method for high-frequency trading (HFT) by integrating three neural network deep learning models, i.e., Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) and Transformer; and we abbreviate the new method to online LGT or O-LGT. The key innovation underlying our method is its efficient storage management, which enables super-fast computing. Specifically, when computing the forecast for the immediate future time, we use only the output calculated from the previous trading data (rather than the previous trading data themselves) together with the current trading data. Thus, the computing involves updating only the current data into the process. We evaluate the performance of O-LGT by analyzing the high-frequency Limit Order Book (LOB) data from the China market. It shows that our model in most cases achieves similar speed with much higher accuracy than the conventional fast supervised learning models for HFT. However, with a slight sacrifice in accuracy, O-LGT is approximately 40 times faster than the existing high-accuracy neural network models for the LOB data in China market.

Keywords: high-frequency limit order book; online fast prediction; hybrid neural network

1. Introduction

More and more investment institutions have entered the trading practice as the financial markets have been significantly growing in recent years. This leads to a rapid increase in the amount of financial time series data generated through high-frequency trading on the financial markets, presenting both opportunities and challenges for researchers to tackle. Our study in this paper will focus on analyzing the time series data of limit order books (LOBs) for high-frequency trading (HFT). LOBs are records of outstanding limit orders maintained by the security specialists who work at the exchange. A limit order is a type of order to buy or sell a security at a specific price or higher. LOBs can be regarded as financial time series that reflect expected price levels for traders. By using a limit order book, traders can specify the exact price at which they want to buy or sell a security such as an asset stock, so that the involved risk can be properly managed and the prospective returns can be maximized. However, LOBs are characterized by low signal-to-noise ratio, non-stationarity and non-linearity (Henrigue et al., 2019), meaning a challenge to effectively and efficiently analyze them.

Stock (more generally, security) price prediction is a key task in analyzing the LOBs data that helps investors develop trading strategies and select investment portfolios that are more likely to produce high returns with low risks. Accurate forecasting of stock prices requires a robust and efficient model. Despite the abundance of research in this field, challenges associated with the speed computing of such models remain. The main objective of our paper is to develop a fast online hybrid neural network model to predict stock prices.

To prepare for this development, a brief review of the current methods for stock price prediction is presented in the following. Overall, most of these methods fall into three categories: statistical parametric models, machine learning techniques, and deep learning approaches.

Regarding the statistical parametric models for stock price prediction, Cenesizoglu et al. (2016) extracted the informative variables that characterize LOBs, so as to establish a vector autoregressive (VAR) model to analyze how various features of the LOBs affect the prices. Mondal et al. (2014) evaluated the accuracy and variability of the stock price forecasts by using autoregressive integrated moving average (ARIMA) model. They employed the model selection criterion AICc to estimate the optimum ARIMA model and also analyzed the impact of altering the time frame of historical data on prediction accuracy. Tran et al. (2017) employed multilinear discriminant analysis (MDA) to forecast large-scale mid-price movements through high-frequency limit order books data. Although statistical parametric models are computationally efficient, they have limitations when applied to complex stock prices data and may not yield the desired results due to their strong dependence on assumptions that are not met by these data.

A data-driven machine learning model typically is not constrained by assumptions, instead, it uses the data themselves to identify patterns and relationships that can inform predictions. Yun et al. (2021) proposed a system for predicting the direction of stock prices movement that emphasizes an enhanced feature engineering process. The system utilizes a hybrid of genetic algorithms and extreme gradient boosting (GA-XGBoost) to optimize the selection of features used in the prediction. Kercheval and Zhang (2015) used a multi-class support vector machine to capture the dynamics of high-frequency LOBs which automatically predicts mid-price movement and other indicators in real time. Previous works in machine learning for stock price prediction have highlighted the importance of extracting relevant features from the underpinning big data for prediction.

Deep learning is a branch of machine learning that uses neural networks each with multiple layers to analyze data. These layers sequentially transform the raw data into informative statistics, allowing the model to extract important features and patterns from the raw data. Convolutional neural network (CNN) is a typical example. Tsantekidis et al. (2017) applied a deep learning approach that forecasts stock prices movement by utilizing CNN. Experiments show that the results of CNN outperform many other machine learning models such as support vector machines. However, compared to other network structures, CNN is relatively unsophisticated and underperforms in analyzing high-frequency trading data.

Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho et al., 2014), have been widely used to predict stock prices. These architectures are well suited for time-series data, such as stock data, because they have the ability to keep previous inputs in their memory, which is important for incorporating the temporal dependencies from the inputs into the prediction process. Moreover, the Transformer (Vaswani et al., 2017) architecture is another type of neural network model that utilizes a self-attention mechanism to weigh the importance of various input sequences. This is particularly useful for stock prediction, as it enables the model to weigh the importance of all financial indicators from the past, such as previous stock prices and volumes themselves.

Hybrid neural networks, which combine different types of neural networks, can have better overall performance in stock prediction because they take advantage of the strengths of their respective architectures. Zhang et al. (2019) proposed deep convolutional neural networks for Limit Order Books (DeepLOB). Three building blocks make up the network architecture of DeepLOB: convolutional layers, parallel inception layers, and an LSTM layer. Zhang and Zohren (2021) proposed DeepAcc for LOBs, which combines DeepLOB with a hardware acceleration mechanism for performing stock prediction. Both DeepLOB and DeepAcc utilize a CNN model as the encoder, which transforms the stock data into a vector. This is followed by a decoder that produces the final output.

While the accuracy of price prediction is undoubtedly a key performance indicator, the computing speed by these deep learning methods are largely ignored in assessing their performance in the literature. In fact, speed is crucial in high-frequency trading to a number of strategies, e.g. cross-market arbitrage and market making. Baron et al. (2019) discovered that variations in relative latency can have a significant impact on the trading performance of HFT firms when they investigated the competition among these firms. Therefore, it is necessary to improve the prediction speed of their methods for investors to obtain more profits without unduly risks.

Motivated by the above review and discussion, we propose an online hybrid neural network method for predicting stock prices based on high-frequency LOBs time series data, where we focus on achieving optimal computing speed while maintaining high prediction accuracy and feasible computing memory. Our proposed method is derived by integrating the three neural network deep learning models LSTM, GRU and Transformer into an online architecture, hence named online LGT or O-LGT. The key innovation underlying O-LGT is its efficient storage management enabling super-fast computing. Specifically, when computing the stock forecast for the immediate future time, we use only the output calculated from the previous trading data (rather than the previous trading data themselves) together with the current trading data. Thus, the computing involves updating only the current data in the process. Details of the method are presented in Section 3.

Comparisons of our proposed method with the currently available stock price prediction methods reviewed above are also performed in this paper. For the China stock market LOBs data that is used in this paper, we find the best of the reviewed methods typically takes at least 2.21 milliseconds of computing time to reach the level of accuracy achieved by the O-LGT method. On the other hand, on the same computer with the same computing power it typically takes O-LGT 0.0579 milliseconds of computing time to reach the same level of accuracy, approximately 40 times faster than the best of the reviewed methods. More details of the comparison are presented in Section 4. Improvement of the computing speed has significant implications for the traders in HFT, because it gives them more time to make decisions and execute orders earlier than their competitors.

This paper is structured as follows. In Section 2, we describe the LOBs data motivating our work in this paper. Next, we present the development of the methods in Section 3, including the problem statement, methods of RNN, LSTM, GRU and Transformer, and the framework of our O-LGT method. Section 4 presents the details and results of our experiments. Finally, Section 5 offers a summary of the paper.

2. Data and Materials

2.1. High Frequency Limit Order Books Data

As introduced in Gould et al. (2013), more than 50% of the global financial markets use limit order books (LOBs) to match sellers and buyers. We first recall some definitions of a limit order book (LOB). A limit order is a contract to buy or sell a limited quantity of shares at a specific price (Gould et al., 2013). Specifically, a sell limit order ensures that the seller sells a specified amount of shares of a stock at a price no less than the specified ask price. Respectively, a buy limit order ensures that the buyer buys a specified amount of shares of a stock for no more than the specified bid price. Accordingly, orders in a LOB can be either ask orders or bid orders.

At a given time t, let $\mathbf{p}_{ask}(t)$ and $\mathbf{v}_{ask}(t)$ represent the column vectors of prices and volumes of all the ask orders; $\mathbf{p}_{bid}(t)$ and $\mathbf{v}_{bid}(t)$ represent the column vectors of prices and volumes of all the bid orders. Also, let $p_{ask}^{(1)}(t)$ be the lowest available sell price in the ask orders and $p_{bid}^{(1)}(t)$ be the highest available buy price in the bid orders. When $p_{ask}^{(1)}(t) < p_{bid}^{(1)}(t)$, the orders are executed and the traded assets are exchanged between the investors on a first-come, first served basis.

Figure 1 gives an illustration of LOBs in trading at a given time stamp, where the five-best priced buy LOB bars and the five-best priced sell LOB bars are sorted according to the price, and the height of each bar represents the volume of the associated LOB. Here the highest bid price $p_{bid}^{(1)}(t)$ is greater than the lowest ask price $p_{ask}^{(1)}(t)$, thus a transaction, of the size equal to the volume of the lowest sell limit

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 February 2023 doi:10.20944/preprints202302.0004.v1

order, is immediately completed between the lowest sell limit order and the highest buy limit order, whereafter the highest buy limit order still has some volume left to be executed because its volume is greater than that of the lowest sell limit order. In cases when multiple sell and buy limit orders partially match, the transactions will be carried out on the first come, first served basis until no more matches exist in the market.

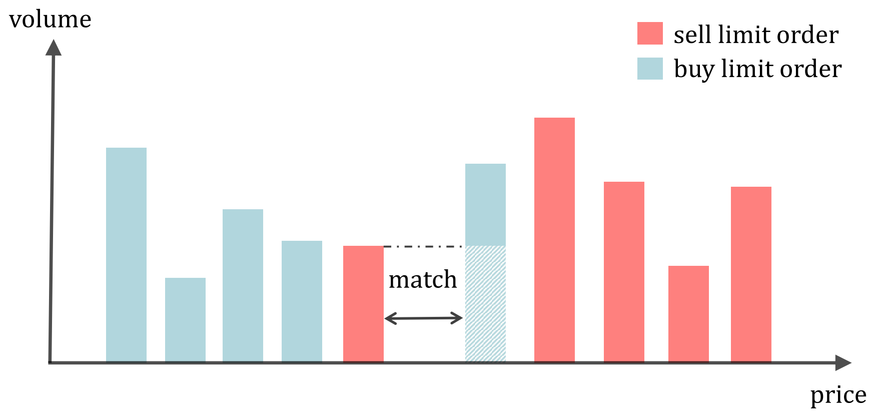

Figure 1. An illustration of LOBs in trading at a given time stamp, where the bid and ask orders are sorted by price. When a bid order is priced higher than an ask order, the two are automatically matched and put into execution.

In HFT markets, the LOBs are traded electronically in high frequency, large numbers, and huge volume. To facilitate investors to receive large profits with low risks from HFT, it is important to provide them with timely information of accurate stock price predictions continuously. Thus, a key task in HFT of LOBs is developing an online stock price prediction model based on the LOBs data.

2.2. LOBs in China Market

The LOB data to be analyzed in this paper comes from the CSI Smallcap 500 Index in the China market. The observed data includes trading records of 100 stocks. These records were updated every 3 seconds for a total of 22 trading days in November 2021. Each trading day has 3 hours and 57 minutes of active trading time, resulting in a total of 4740 records per day.

Each trading record can be summarized into 26 features which will be used in this paper. Specifically, these features include the highest five bid prices $p_{bid}^{(1)}(t), \cdots, p_{bid}^{(5)}(t)$ and the corresponding LOB volumes $v_{bid}^{(1)}(t), \cdots, v_{bid}^{(5)}(t)$; the lowest five ask prices $p_{ask}^{(1)}(t), \cdots, p_{ask}^{(5)}(t)$ and the corresponding LOB volumes $v_{ask}^{(1)}(t), \cdots, v_{ask}^{(5)}(t)$; average transaction price over the last three seconds $p_{tra}^{avg}(t)$; total ask volumes $v_{ask}^{(all)}(t)$; total bid volumes $v_{bid}^{(all)}(t)$; average ask price $p_{ask}^{avg}(t)$; average bid price $p_{bid}^{avg}(t)$; the latest transaction price $p_{last}(t)$.

3. Methods

3.1. Problem Statement

The problem aimed to be tackled in this paper is to develop an online hybrid neural network model for continuous prediction of stock prices based on high-frequency LOBs data. Let $p(T)$ be the price of a stock at time T, with $T = 1, 2, \cdots$ denoting the number of time units passed from the beginning of trading on each day. For the China LOBs data, the time unit is 3 seconds. Predicting $p(T)$ is equivalent to predicting the target variable y_T, defined as $y_T = \frac{p(T)-p(T-h)}{p(T-h)}$, which represents the percentage change in the stock price between time T and h units of time earlier. In China HFT market one typically predicts y_{T+h} at time T with $h = 100$, i.e., predicts the price 5 minutes forward. Denote $\mathbf{x}_t$ as the $J = 26$ features of the China LOBs data recorded at time t as defined in section 2.2. Then the prediction $\hat{y}_T$ of y_T by the LOB features recorded in the previous s time steps can be generically formulated as the following

$$\hat{y}_T = \hat{F}\left(\mathbf{x}_{T-1}, \mathbf{x}_{T-2}, \cdots, \mathbf{x}_{T-s}\right) \tag{1}$$

where $F(\cdot)$ denotes a generic neural network and $\hat{F}(\cdot)$ is an estimate of $F(\cdot)$ obtained from the training data. The best predictions $\hat{y}_T$ across all values of T are to be computed by minimizing $D(y, \hat{y})$, where $D(\cdot, \cdot)$ is a customized discrepancy function that measures the proximity of an estimate to its actual value.

We will develop an online hybrid neural network method to formulate and optimally estimate $F(\cdot)$. This method is named O-LGT since it is in the form of a general recurrent neural network (RNN) containing multiple latent layers that are respectively specified by the long short-term memory (LSTM) model, the gated recurrent unit (GRU) model, and the Transformer model in sequence. Also, the method is implemented in an online way; i.e., is updated once the time moves forward by one unit. In order to give a detailed description of O-LGT, we first review all of its layers in the following.

3.2. Recurrent Neural Network (RNN)

First, recall that Recurrent Neural Networks (RNNs) are a type of neural network that are designed to process sequential data, such as time series data. An RNN contains a hidden compartment that inputs information on the data from both the previous and the current time steps, and outputs predictions for future time steps. This hidden compartment is updated once at each new time step, by passing the input through to generate the output.

The structure of RNN is cyclical, meaning that the same computation is done at each step using the same parameters, which is why it is called recurrent. The architecture of RNNs can be unrolled to show the sequence of computations that occur over time.

Figure 2 is a typical RNN structure diagram, where $\mathbf{h}_t$, the output vector of hidden layer at time t depends not only on the input vector $\mathbf{x}_t$ at time t, but also on $\mathbf{h}_{t-1}$. The $\mathbf{h}_t$ is calculated using an activation function $f(\cdot)$ as follows:

$$\mathbf{h}_t = f\left(U\mathbf{x}_t + W\mathbf{h}_{t-1}\right) \tag{2}$$

where U is the input layer weights matrix and W represents the hidden layer weight matrix. The RNN is initialized at time $t = 1$ with

$$\mathbf{h}_1 = f\left(U\mathbf{x}_1 + W\mathbf{h}_0\right) \quad \text{and} \quad \mathbf{y}_1 = g\left(V\mathbf{h}_1\right) \tag{3}$$

where $g(\cdot)$ is the activation function for the output layer and V is the associated weight matrix, then proceeds with the following operations at each time stamp $t = 2, 3, \cdots$:

$$\mathbf{h}_t = f\left(U\mathbf{x}_t + W\mathbf{h}_{t-1}\right) \quad \text{and} \quad \mathbf{y}_t = g\left(V, \mathbf{h}_t\right) = g\left(V\mathbf{h}_t\right) \tag{4}$$

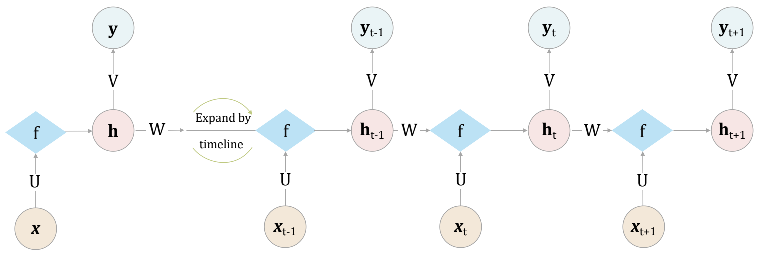

Figure 2. Structure of a recurrent neural network. $\mathbf{x}_t$ is the input at time t; $\mathbf{h}_t$ denotes the output of hidden layer at time t; $\mathbf{y}_t$ is final output at time t; U represents the input layer weight matrix; W represents the hidden layer weight matrix, and V represents the output layer weight matrix; $f(\cdot)$ is the activation function.

3.3. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU)

Hochreiter and Schmidhuber (1997) introduced the Long Short-Term Memory (LSTM) architecture to address the problem of vanishing gradients in traditional Recurrent Neural Networks (RNNs) when trying to model long-term dependencies in sequential data.

The core of LSTM is a cell consisting of input gate $\mathbf{i}_t$, omission gate $\mathbf{f}_t$, interim cell state $\widetilde{\mathbf{c}}_t$, and interim output gate $\mathbf{o}_t$, normally in a form of column vectors as shown in the left panel of Figure 3. The omission gate $\mathbf{f}_t$ determines the information to be directly passed to the cell state output $\mathbf{c}_t$, the input gate $\mathbf{i}_t$ determines the information to be further used in the cell together with the interim cell state $\widetilde{\mathbf{c}}_t$, and the interim output gate $\mathbf{o}_t$ together with the cell state output $\mathbf{c}_t$ gives the hidden layer output $\mathbf{u}_t$.

The "long" in LSTM stands for the capability of the model to retain information for a long period of time. The memory cells in LSTMs, which can keep information for multiple time steps, enable the model to effectively identify long-term dependencies in sequential data. The "short" in LSTM stands for the model's ability to discard irrelevant information promptly. An LSTM model uses the omission gate and input gate to appropriately store and retrieve selected information from the memory cells.

At each time t, an LSTM model with hidden layer dimension q_L loads the $J \times 1$ feature vector $\mathbf{x}_t$, the $q_L \times 1$ hidden layer output vector $\mathbf{u}_{t-1}$ at the previous time stamp, and the $q_L \times 1$ cell state vector $\mathbf{c}_{t-1}$ at the previous time stamp as the input; then uses certain activation functions to generate $[\mathbf{f}_t, \mathbf{i}_t, \widetilde{\mathbf{c}}_t, \mathbf{o}_t]$ which is a $q_L \times 4$ matrix; and finally outputs the updates $\mathbf{u}_t$ and $\mathbf{c}_t$. An update in LSTM at time t can be expressed as follows:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(W_i \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_{t-1} \end{bmatrix} + \mathbf{b}_i\right) \\
\widetilde{\mathbf{c}}_t &= \tanh\left(W_c \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_{t-1} \end{bmatrix} + \mathbf{b}_c\right) \\
\mathbf{f}_t &= \sigma\left(W_f \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_{t-1} \end{bmatrix} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(W_o \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_{t-1} \end{bmatrix} + \mathbf{b}_o\right) \\
\mathbf{c}_t &= \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} \\
\mathbf{u}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)
\end{aligned} \tag{5}$$

where W_i, W_f, W_o and W_c are the $q_L \times (J + q_L)$ weight matrices of the input gate, the omission gate, the output gate, and the cell state respectively; $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$ and $\mathbf{b}_c$ are the $q_L \times 1$ bias vector parameters of the input gate, the omission gate, the output gate, and the cell state respectively. Also, $\odot$ is the Hadamard product operation. Moreover, $\sigma(\cdot)$ and $\tanh(\cdot)$ are activation functions defined as

$$\sigma(x) = \frac{1}{1+\exp(-x)} \quad \text{and} \quad \tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)} \tag{6}$$

GRU model is a simplified version of the LSTM model and was introduced by Cho et al. (2014). Compared with LSTM, GRU has fewer parameters and is computationally less expensive to be trained, while still capable of capturing long-term dependencies in sequential data and being robust against overfitting. The main difference between LSTM and GRU is that GRU combines the omission and input gates of LSTMs into a single update gate $\mathbf{z}_t$. The GRU structure is depicted in the right panel of Figure 3. The core of GRU is a cell consisting of reset gate $\mathbf{r}_t$, update gate $\mathbf{z}_t$, and interim output $\tilde{\mathbf{v}}_t$.

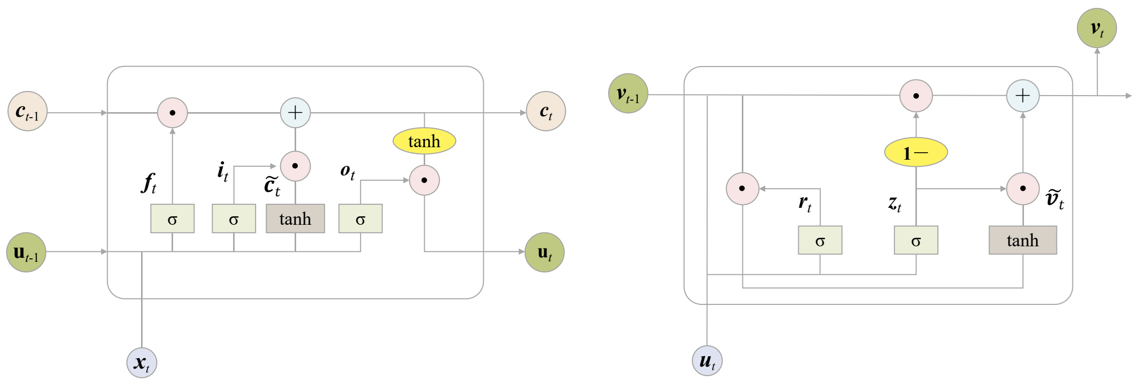

Figure 3. The **left** panel shows the structure of long short-term memory, and the **right** panel shows the structure of gated recurrent unit.

The "Gated" in GRU refers to the use of gates to control the flow of information in and out of the hidden layer. "Recurrent" refers to the fact that GRU is a type of Recurrent Neural Network (RNN) that processes sequential data by passing information from one time step to the next.

At each time t, a GRU model with hidden layer dimension q_G loads the $q_L \times 1$ LTSM hidden layer output $\mathbf{u}_t$ and the $q_G \times 1$ GRU output vector $\mathbf{v}_{t-1}$ at the previous time stamp as the input; then uses the activation functions $\sigma(\cdot)$ and $\tanh(\cdot)$ to compute $[\mathbf{r}_t, \mathbf{z}_t, \tilde{\mathbf{v}}_t]$ which is a $q_G \times 3$ matrix; and finally generates the updated GRU output vector $\mathbf{v}_t$. An update in GRU at time t can be expressed as follows:

$$\begin{aligned}
\mathbf{v}_t &= (1-\mathbf{z}_t) \odot \tilde{\mathbf{v}}_t + \mathbf{z}_t \odot \mathbf{v}_{t-1} \\
\tilde{\mathbf{v}}_t &= \tanh\left(W_v \begin{bmatrix} \mathbf{u}_t \\ \mathbf{r}_t \odot \mathbf{v}_{t-1} \end{bmatrix} + \mathbf{b}_v \right) \\
\mathbf{r}_t &= \sigma\left(W_r \begin{bmatrix} \mathbf{u}_t \\ \mathbf{v}_{t-1} \end{bmatrix} + \mathbf{b}_r \right) \\
\mathbf{z}_t &= \sigma\left(W_z \begin{bmatrix} \mathbf{u}_t \\ \mathbf{v}_{t-1} \end{bmatrix} + \mathbf{b}_z \right)
\end{aligned} \tag{7}$$

where W_v, W_z and W_r are the $q_G \times (q_L + q_G)$ weight matrices of the interim GRU output, the update gate and the reset gate respectively; and $\mathbf{b}_v$, $\mathbf{b}_r$, and $\mathbf{b}_z$ are the $q_G \times 1$ bias vector parameters of the interim GRU output, the reset gate and the update gate respectively.

Note that the reset gate $\mathbf{r}_t$ is used to control the extent to which the output information of the previous time moment is ignored. Typically, the smaller the value of the reset gate, the more likely it is ignored. In addition, a larger value of the update gate $\mathbf{z}_t$ indicates that the neural unit at the current time moment is less influenced by the output information of the neural unit from the previous time moment.

3.4. Transformer

After being processed by LSTM and GRU, the multi-headed transformer aids in extracting useful information regarding the interactions in outputs between various time steps.

There are significant differences between the Transformer (Vaswani et al., 2017) and the traditional RNN model, in that the Attention mechanism in the former completely determines the structure of the entire network. Attention allows the model to focus on specific parts of the input by assigning different weights to different positions in the input sequence. This is in contrast to the traditional RNNs which use the same weights for all positions in the input sequence.

The Transformer model uses the encoder-decoder architecture that is most commonly used in Neuro-linguistic programming. This architecture provides an effective way to handle long sequence data (Bahdanau, 2014). In the Transformer model, the encoder has four layers: The first layer uses a Multi-Head Attention mechanism (Multi-Head Attention) to assign multiple sets of different attention weights to the model for extending the model's ability to focus on different locations, thus capturing richer information than otherwise. The second layer is the summation and normalization layer, where the summation, also named the Residual Connection, adds the interim output of the layer to its input before being normalized to produce the layer's final output. The second layer means to pass the information from the previous layer to the next layer without differences to solve the gradient disappearance problem more quickly. The third layer is a Feed Forward Neural Network (FNN) layer. The fourth layer then goes through another summation and normalization layer to generate the intermediate semantic coding vector and transmit it to the decoder. The decoder has six layers, similar to the encoder structure, but the first layer is a multi-headed attention layer with MASK (masking) operation, because at output time t, the information at time $t + 1$ is not available, so the output of the decoder needs to be shifted right and the subsequent items are masked for prediction, and finally the decoder then goes through linear regression and Softmax layer to output the final prediction result of the decoder. Figure 4 displays the block diagram for the Transformer model.

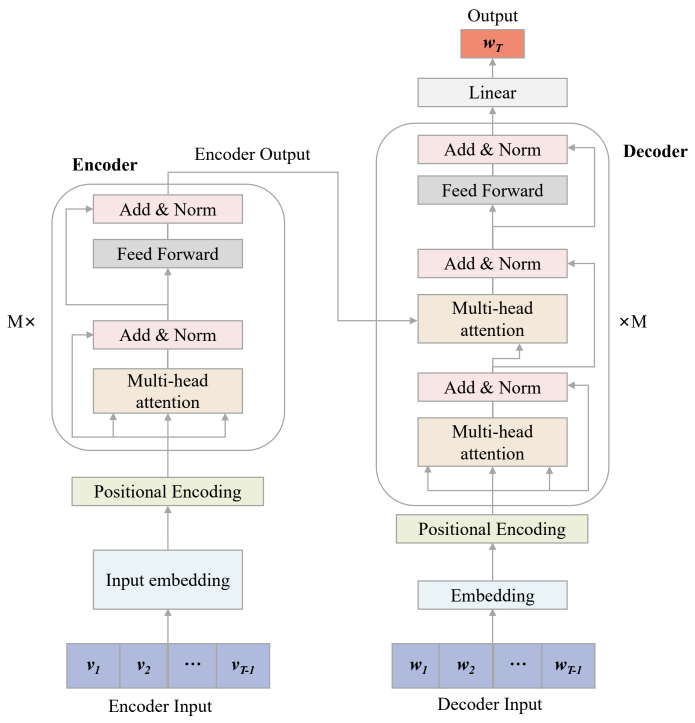

Figure 4. Illustration of a Transformer. The dimension of $\mathbf{v}_i$ is the input dimension of the transformer and the dimension of $\mathbf{w}_i$ is the hidden layer dimension of the transformer.

3.5. Online LGT (O-LGT)

Having reviewed all necessary RNNs for developing our new stock price prediction method O-LGT, we are ready to present the details of this development in this section. We first give a brief description of the sweep matrix operator, the core computing engine underpinning O-LGT. We then describe the framework of O-LGT, followed by its implementation in practice. Finally we describe input data standardization and transformation before loading them to execute O-LGT.

3.5.1. Sweep Operator

Computations involved in all neural network layers in O-LGT are essential for solving weighted regression normal equations $X^T W X \hat{\beta} = X^T W y$. Sweep matrix operator (Beaton, 1964) gives a very efficient method to solve these normal equations. Following Goodnight (1979), a square matrix $M = (m_{ij})$ is said to have been swept on the kth row and column (or kth pivotal element) when it has been transformed into a matrix $N = (n_{ij})$ such that

$$\begin{aligned} n_{kk} &= 1/m_{kk} \\ n_{ik} &= -m_{ik}/m_{kk} \quad i \neq k \\ n_{kj} &= m_{kj}/m_{kk} \quad j \neq k \\ n_{ij} &= m_{ij} - m_{ik}m_{kj}/m_{kk} \quad i, j \neq k. \end{aligned} \tag{8}$$

By a sweep operation on each pivotal element of M, each element of M is updated essentially by one division operation. By applying sweep operations on all pivotal elements of $X^T W X$, the normal equation $X^T W X \hat{\beta} = X^T W y$ can be solved in approximately $O(2||X^T W X||_0^2)$ divisions, with $||X^T W X||_0$ being the number of elements in matrix $X^T W X$, which is the same computing complexity as that involved in the classical Gauss-Jordan elimination operator. However, when an extra column of data is augmented to X, the resultant new normal equation can be solved by updating the process of solving the previous normal equation by applying just one more sweep operation on the new pivotal element, which takes only $O(||X^T W X||_0)$ extra divisions. This suggests the sweep operation is a much faster algorithm than the classical Gauss-Jordan elimination algorithm in online computing where the new data feeds into the process sequentially. This is the major reason our proposed O-LGT method uses the sweep operator in its core computing engine.

3.5.2. O-LGT Framework

The O-LGT model combines LSTM, GRU, and Transformer as three sequential layers into a composite RNN to predict stock prices in HFT. The LSTM layer is for capturing important information from the input data to return accurate LSTM output. The GRU layer takes the LSTM output as the input which is then processed in a condensed way to prevent overfitting. Then the Transformer layer takes the GRU layer output as the input which is processed in such a way that the interaction effects between different time steps are incorporated into the output at a more granular level. Finally, the predictions are made by concatenating the Transformer output with its time length information and feeding the result through a linear regression layer. As seen from the previous sub-sections the structure of each neural network layer in O-LGT is easy to understand, train, and compute on its own, but our innovative implementation setting for O-LGT, which will be detailed in section 3.5.3, has significantly accelerated the computation of O-LGT without compromising its accuracy. The model architecture schematic of O-LGT is displayed in Figure 5 and is explained in the following.

At the initial step, the model inputs X_T, the matrix of observed features of the LOB data for the previous T moments, into an LSTM layer, and the output of this layer, U_T, is stored. This output is then input into the next layer, a GRU layer, and the output of this layer, V_T, is stored. The output of the GRU layer is then fed into a Transformer layer, and the output, $\mathbf{w}_T$, is stored. This output and time length information are concatenated into the final linear regression layer to make the final predictions.

At the acceleration steps, the O-LGT model uses a similar process but with the added input of the previous moment's output for each layer. The input at time t and the output of the LSTM layer from the previous moment, $\mathbf{u}_{t-1}$, are input into the LSTM layer, the output of which, $\mathbf{u}_t$, is stored. The output of the LSTM layer and the GRU layer output from the previous time moment, $\mathbf{v}_{t-1}$, are input into the GRU layer, with the output of which, $\mathbf{v}_t$, being stored. Lastly, the output of the GRU layer and Transformer layer output from the previous moment, $\mathbf{w}_{t-1}$, are input into the Transformer layer, the output of which, $\mathbf{w}_t$, is stored. This output and time length information are concatenated into the final linear regression layer to make the latest prediction.

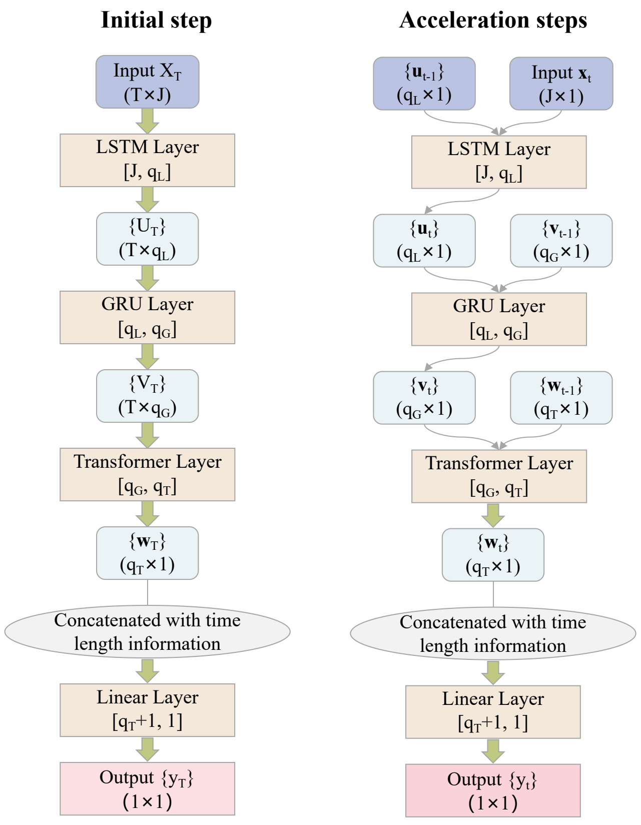

Figure 5. O-LGT model architecture schematic. The left panel shows the details of the prediction steps for y_T at initial step and the right panel shows the details of the prediction steps for y_t at acceleration steps. T represents the length of time for the input at initial step; J represents the input dimension; q_L, q_G, q_T represent the hidden layer dimensions of the LSTM, GRU and Transformer, respectively. The values of $[\cdot, \cdot]$ represent input dimensions and output dimensions. The values of $(\cdot \times \cdot)$ represent dimensions of a matrix and a vector.

3.5.3. Experimental Design for Implementation

In an HFT market, the LOBs data arrives at a very high frequency (every 3 seconds in the case of China HFT market). It is both computationally and logistically impractical to predict stock prices for the time steps immediately after the current time step when the latest LOBs data arrives. Thus, we propose to predict the prices (in terms of price percentage change) by O-LGT for time step $T + h$ when standing at the current time step T, where $h = 100$ (i.e., 5 minutes) is chosen for our China market case study. On the other hand, there is no need to use the LOBs data from all past time steps until the current time step T to predict the prices for time step $T + h$, because the stock price dynamics manifest an LSTM behavior. This behavior is fully utilized by our O-LGT framework in that, when the current time step is between T and $T + b$ inclusive, we use only the LOBs data from time step T back to time step T_s to make predictions for time steps $T+h$ until $T+h+b$. In this way, the required feature input data for processing O-LGT at a time step between T and $T+b$ is of a time length not more than $s+b+1$, thus taking a very limited amount of computer memory, resulting in a significant acceleration of the predicting process by O-LGT. We term the implementation procedure just described for O-LGT the moving window based prediction technique with the back, current, and future window sizes being s, b, and h, respectively. For the China HFT market case study, we choose $s = 99$ (4 minutes 57 seconds), $b = 19$ (57 seconds), and $h = 100$ (5 minutes). This moving window technique is illustrated in Figure 6, where the blue box represents the input features of the model and the red box represents the output of the model.

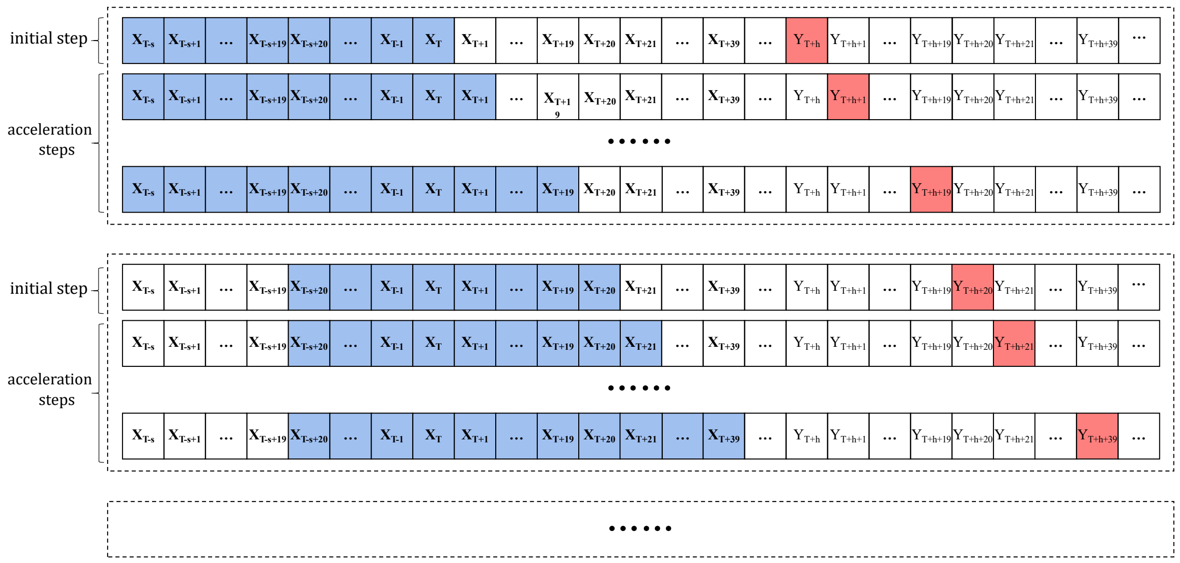

Figure 6. Adjusted moving window method process. Here, $s = 99$, $b = 19$, and $h = 100$. Observations of LOB data are updated every 3 seconds. The operation in the dashed box is updated every 20 time units (1 minute). The dashed box indicates an acceleration process, where $\mathbf{x}_t$ indicates the feature information at time t and Y_{T+h} represents the predicted stock price percentage change at time $T+h$. The first line in the first dashed box indicates the stock price prediction Y_{T+h} at time $T+h$ is obtained by using the feature information from time $T-s$ to time T. The last line in the first dashed box indicates the stock price prediction Y_{T+h+19} at time $T+h+19$ is obtained by using the feature information from time $T-s$ to time $T+19$.

Our O-LGT method also ensures high prediction accuracy as the other RNN methods. Denote $\mathbf{x}_{c:d}$ as the sequence of input feature data from time c to time d for a stock. Given $\mathbf{x}_{1:T}$ of arbitrary time length T and the future window size h, the online sequential way that the target variable y_{T+h} is predicted by the O-LGT model suggests y_T can be formulated as follows:

$$y_{T+h} = F\left(\mathbf{x}_{(T-s):T}, \tilde{\mathbf{x}}_{T-s}\right), \tag{9}$$

where $F(\cdot,\cdot)$ is a generic function, $\tilde{\mathbf{x}}_{T-s}$ is the summary information obtained from executing O-LGT at time $T-s$, and s is the back window size. Here, the back window size s cannot be too small, otherwise, the result is easy to underfit; and s cannot be too large, otherwise, there is not enough space for storage. Typically, we select the appropriate s according to the performance of the training data. It can be observed that the O-LGT processing of the input data sequence follows a Markov dependence pattern. Only the latest input data block $\mathbf{x}_{(T-s):T}$ at time T and the previously aggregated information $\tilde{\mathbf{x}}_{T-s}$ are related to the prediction $\mathbf{y}_{T+h}$ at the current time moment T. Hence it is reasonable to conclude that an optimal value of s can be found based on the training data so that the O-LGT algorithm with this optimal s value will produce highly accurate predictions. However, we acknowledge that the optimal s value determined this way may be too large to impact the computing complexity and speed. Fine-tuning the back window size s based on trader's experience is still important and even necessary.

3.5.4. Data Standardization and Transformation

Recall that an RNN is characterized by the fact that all the data at different moments share the same structure and coefficient specifications. Therefore, when the model inputs are of the same length, the data will follow the same distribution at $T = 1, 2, \cdots$. However, for some variables with cumulative effects over time, such as current prices, the cumulative effects result in different variances. For example, the price at moment $t = 95$ is different from that at moment $t = 120$. Therefore, for variables with cumulative time effects, we transform the input from the absolute value to the percentage change of the current moment with respect to the previous moment. Therefore, the transformed variable will follow exactly the same distribution regardless of the time length. This treatment is more consistent with RNNs sharing the same network structure at different time lengths.

4. Experiment

In this paper, several experiments are conducted on the CSI Smallcap 500 Index (CSI-500) in China to demonstrate the performance of the O-LGT model in stock market forecasting.

4.1. Data Pre-processing for CSI-500

The dataset and the involved features are described in section 2. Since stock prices are influenced by various factors that are not known to us, the collected data appears to contain a large amount of noise and fluctuations that are actually determined by these unknown factors. If the original stock data is directly entered into the modeling and analysis process without data pre-processing, it may reduce the accuracy of the results significantly. To increase the prediction accuracy, an appropriate data pre-processing is done as shown below.

$$\widetilde{P}(t) = \frac{p(t) - p(t-1)}{p(t-1)} \tag{10}$$

$$\widetilde{p_{\text{ask}^{(k)}}}(t) = p_{\text{ask}}^{(k)}(t) - p(t), \quad k = 1, 2, \ldots, 5 \tag{11}$$

$$\widetilde{p_{\text{bid}^{(k)}}}(t) = p_{\text{bid}}^{(k)}(t) - p(t), \quad k = 1, 2, \ldots 5 \tag{12}$$

$$\widetilde{v_{\text{ask}}}(t) = \frac{v_{\text{ask}}(t)}{v_0} \tag{13}$$

$$\widetilde{v_{\text{bid}}}(t) = \frac{v_{\text{bid}}(t)}{v_0} \tag{14}$$

To prevent excessive data volatility, equations 10 to 14 are also processed, with v_0 denoting the day's opening volume on the stock market. This keeps the data essentially stationary and maintains the same distribution of the same variable.

4.2. Experiment Setting

In the experiments, computations by the O-LGT model were performed under the python environment and the PyTorch framework. The Adam optimization method was used with a learning rate of 0.001 and an exponential linear decay of 0.95 after each epoch. The model's parameters were updated based on the gradients of the mean square error loss function, and the model was trained for 100 epochs using the training dataset. Table 1 lists the software and hardware configurations used for the experiments, while Table 2 lists the values of the tuning-parameters used in the model.

Table 1. Hardware and Software Configurations for Experiments.

Item	Configuration
Python Version	Python 3.9
Pytoch Version	1.10.2
CPU	i7-7500U 2.70GHz
RAM	16G

Table 2. Tuning-parameter Specification for Experiments.

Item	Tuning-parameter
Optimization	Adam
Initial Learning Rate	0.001
Exponential Linear Decay	0.95
Epoch Number	100

4.3. Prediction Error Evaluation

The best prediction for stock prices is achieved by minimizing the loss function which calculates the total differences between the predicted and the true price values. The resultant minimum loss provides a measure of prediction accuracy. Commonly used loss functions include R-squared, mean squared error (MSE), and mean absolute error (MAE), which are defined below

- R-squared (R^2):
$$r^2 = \frac{\sum_{i=1}^{N} r_i^2}{N}, \quad r_i^2 = 1 - \frac{\sum_{t=1}^{n}(\hat{y}_{it} - y_{it})^2}{\sum_{t=1}^{n} y_{it}^2} \tag{15}$$
- Mean Squared Error (MSE):
$$L_{mse} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{n}(y_{it} - \hat{y}_{it})^2 \tag{16}$$
- Mean Absolute Error (MAE):
$$L_{mae} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{n}|y_{it} - \hat{y_{it}}| \tag{17}$$

Here $\hat{y}_{it}$ denotes the predicted return of rate for stock i at time t; y_{it} denotes the true return of rate for stock i at time t; N is the number of stocks; n is the number of time steps; r_i^2 is the R-squared for stock i; and r^2 is the R-squared for the model. It is easy to see that a small MSE or MAE value corresponds to the high accuracy of the prediction, while a high value of R-squared corresponds to the high accuracy of the prediction. A stock price forecasting model with an R-squared value greater than 0.01 is normally considered to be reliable.

4.4. Implementation Details

In this section, we present details of applying the O-LGT approach for analyzing the China LOBs data. The dataset includes 100 stocks from the CSI Smallcap 500 Index market for a total of one month. Per section 3.1, we set $y_T = \frac{p(T)-p(T-h)}{p(T-h)}$, where $h = 100$ and $p(T)$ represents the price of a stock at time T. Also recall that $\{\mathbf{x}_t, t = 1, \cdots, T\}$ is a J-dimensional vector time series of the LOBs features with $\mathbf{x}_t = \{x_{1t}, x_{2t}, \cdots, x_{Jt}\}^\top$ and $J = 26$.

The left panel of Figure 7 gives a flow chart of the modeling and prediction process for y_T at time T for the initial step, while the right panel is for the acceleration steps, where the back, current and future window size of the associated moving window method are $s = 99$, $b = 19$, and $h = 100$, respectively.

This means, at the initial step when time is at T, $X_T = [\mathbf{x}_{T-99}, \cdots, \mathbf{x}_T]^\top$ — a 100×26 matrix of feature observations in all previous 100 time moments — is the input data we need for O-LGT. The LSTM layer has its input and output both being 20-dimensional. The GRU layer and the multi-head attention Transformer layer are also 20 dimensional for both their input and output. The output of the Transformer layer $\mathbf{w}_T$ and the information with the length of time are concatenated together, being of dimension 21, which is then loaded as the input to the linear layer. The final output is the prediction of the stock price percentage change value y_{T+100}.

At the acceleration steps when $T < t < T + 20$, $\mathbf{x}_t$ has $J = 26$ features in time t; $\mathbf{u}_{t-1}$, $\mathbf{v}_{t-1}$, and $\mathbf{w}_{t-1}$ are 20×1 vectors at time $t - 1$; $\mathbf{u}_t$, $\mathbf{v}_t$, and $\mathbf{w}_t$ are 20×1 vectors at time t. The dimensions of the inputs and outputs are the same for each layer and the final output is the prediction of the stock price percentage change value y_{t+100}.

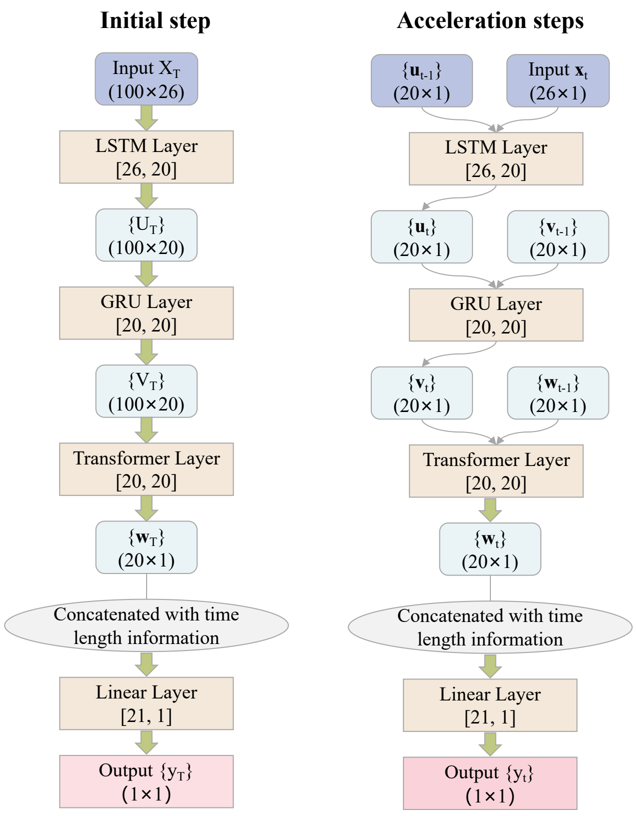

Figure 7. The O-LGT structure schematic for analyzing the China LOBs data. The left panel shows the details of the prediction steps for y_T at initial step and the right panel shows the details of the prediction steps for y_t at acceleration steps.

Regarding the input of O-LGT, recall that we have 4740 time steps for each stock every trading day and there are 22 trading days in November 2021 in China HFT market. LOB information from every segment of 100 (i.e. $s = 99$) plus up to 19 (i.e. $b = 19$) consecutive time steps will be used as a single input, and the rate of return after 5 minutes (i.e. $h = 100$) is the corresponding output. For prediction at each time step $t \in [T, T + b]$, the feature data $\mathbf{x}_{(T-s):t}$ is used as the input, where T is a sharp minute time step (e.g. 9h:50m:00s:am sharp), $b = 19$ (57 seconds), and $s = 99$ (4 minutes 57 seconds). Thus, we have around 4610 consecutive and overlapping segments of input data of form $\mathbf{x}_{(T-s):t}$ on every trading day. Instead of testing the performance of O-LGT on all these 4610 segments of input data on each trading day (which is referred to as fixed testing), we can do it on a stratified random sample from these 4610 segments. For example, one such random sample can be obtained by first partitioning the 4610 segments into 461 consecutive sections and then randomly choosing a segment from each section. This testing based on random sampling is referred to as random testing. A possible advantage of using random testing is the reduced auto correlations in the random sample.

Instead of using a fixed back window size s for training the model, one could use a randomly selected s value at each updating step for the training. For example, s can be randomly selected from the interval $[89, 109]$. Selecting s at random is referred to as the random training.

4.5. Experiment Results

In this section, we report the results of experiments to demonstrate the performance of our O-LGT model. Table 3 lists the results of the three comparison experiments specified by fixed/random training/testing combinations. Table 4 shows the three error measures of prediction accuracy for the three relevant models (with random training/testing setting if applicable) that are available for handling missing values in the input. Table 5 shows the prediction error measures and the spent computing time for six relevant models (with random training/testing setting if possible) that can handle scenarios without missing values.

First, to confirm the validity of the experimental design, we carried out three comparison experiments:

- The moving window for model training has fixed back section size $s = 99$. The model testing is performed on all 4610 input segments.
- The moving window for model training has fixed back section size $s = 99$, while the model testing is performed on a stratified random sample of 461 input segments.
- The moving window for model training has its back section size s being chosen from $[89, 109]$ at random, while the model testing is performed on a stratified random sample of 461 input segments.

Table 3 demonstrates that the performances of the O-LGT method are similar to each other under the three experiments. This verifies the validity of our design.

Table 3. Results of the three comparison experiments to verify the validity of the model design. Values in () represent standard errors. ↑ means the higher the value, the better the model performs. ↓ means the lower the value, the better the model performs.

Methods	$100 \times r^2 \uparrow$	$100\times$ **MSE** ↓	$100\times$ **MAE** ↓
Training fixed -> Test fixed	2.71 (0.0171)	0.0291 (0.00217)	0.0943 (0.00454)
Training fixed-> Test random	1.68(0.0143)	0.0283(0.00210)	0.0910(0.00444)
Training random-> Test random	2.65 (0.0165)	0.0291 (0.00236)	0.0948 (0.00468)

Next, we considered the scenario with missing values. Specifically, values at five time steps in each segment of input data are made missing at random. Our benchmark models are the linear model with faster computation and the GRU with relatively high accuracy. Table 4 shows that when there are missing values, the prediction performance of LGT and O-LGT is much better than that of Linear. Comparing LGT and O-LGT, we find that the r^2 of O-LGT is higher than that of LGT, but there is no significant difference between MSE and MAE. The superiority of O-LGT in the presence of missing values is determined by its design features. This is because O-LGT is more relax in the choice of input length, which makes the prediction accuracy unaffected even in the presence of missing values.

Table 4. Benchmark models comparison in missing value scenarios. Values in () represent standard errors. ↑ means the higher the value, the better the model performs. ↓ means the lower the value, the better the model performs.

Methods	$100 \times r^2 \uparrow$	$100\times$ **MSE** ↓	$100\times$ **MAE** ↓
Linear	0.10 (0.0105)	0.124 (0.0168)	0.229 (0.129)
LGT	2.44 (0.0163)	0.0292 (0.00218)	**0.0944 (0.00454)**
O-LGT	**2.65 (0.0165)**	**0.0291 (0.00236)**	0.0948 (0.00468)

In the absence of missing values, we consider comparing the O-LGT with not only the traditional models mentioned above, i.e., linear model and GRU, but also the DeepLOB and DeepAcc models reviewed in Section 1, which are the two powerful hybrid models with better overall performance. Table 5 demonstrates that O-LGT and LGT have clear advantages over the other comparative models in terms of prediction accuracy. In particular, the linear model and the XGBoost model have small r^2 values, indicating their prediction performance is very poor. In terms of the spent computing time, O-LGT is about 38 times faster than LGT, about 64 times faster than Deeplob, and about 12 times faster than DeepAcc. This shows that our O-LGT can significantly speed up prediction while maintaining prediction accuracy, which is very beneficial for early risk assessment in the stock market.

Table 5. Performance comparison of different models with no missing values. Values in () represent standard errors. ↑ means the higher the value, the better the model performs. ↓ means the lower the value, the better the model performs.

Methods	$100 \times r^2 \uparrow$	100× **MSE** ↓	100× **MAE** ↓	**Time (millisecond)**
Linear	0.15 (0.0111)	0.109 (0.0135)	0.203 (0.0118)	**0.0352**
XGBoost	0.44 (0.097)	0.0699 (0.00778)	0.153 (0.00754)	0.704
DeepLOB	2.37 (0.0210)	0.0301 (0.00219)	0.0945 (0.00457)	3.68
DeepAcc	1.76 (0.0158)	0.0316 (0.00217)	**0.0930 (0.00454)**	0.695
LGT	**2.71 (0.0171)**	**0.0291 (0.00217)**	0.0943 (0.00454)	2.21
O-LGT	**2.65 (0.0165)**	**0.0291 (0.00236)**	0.0948 (0.00468)	**0.0579**

5. Conclusion

This paper developed an online hybrid recurrent neural network model (O-LGT) for LOBs prices analysis in an effort to fast and accurately predict the stock price fluctuations in HFT environment. Three recurrent neural networks: LSTM, GRU and Transformer, are used as the three consecutive layers in O-LGT to extract the information of the features at different stages. We performed experimental studies for the CSI-500 dataset under three setups: Setting I, Setting II, and Setting III. The results are summarized in the following.

- Setting I simulated three comparative experiments specified by the fixed/random training/testing combinations, and the results confirms the validity of the experimental design.
- Setting II simulates a missing value scenario, under which O-LGT is compared with LGT and the linear model.

  The experiment demonstrates that O-LGT is much more accurate than the linear model, and the r^2 value of O-LGT is 8.61% higher than that of LGT, indicating that the random training and testing for O-LGT has a negligible impact on the prediction accuracy in the presence of missing values in the input data.
- Setting III is a scenario with no missing values in the input data. Under this scenario, O-LGT is compared with XGBoost, DeepLOB, and DeepAcc, in addition to the linear model and LGT.

  First, compared to the linear model and XGBoost, the prediction accuracy of O-LGTl is significantly higher than that of these two classical models. Compared to the network integration models DeepLOB and DeepAcc, the r^2 value of O-LGT increased by 11.81% and 50.57%, respectively. It is 3.70 percent less than the r^2 value of LGT. Although the experimental design used in O-LGT model results in a slight loss in prediction accuracy when compared to LGT, O-LGT is computationally substantially faster than the network integration models that have a relatively high accuracy rate. Compared with the network integration models LGT, DeepLOB and DeepAcc, O-LGT is 38 times, 64 times and 12 times faster, respectively.

In conclusion, our O-LGT model has capacity to quickly and accurately predict stock price in high-frequency trading markets.

Author Contributions: Conceptualization, C.L. and G.Q.; methodology, C.L., L.S. and G.Q.; software, C.L. and L.S.; validation, G.Q. and C.L.; formal analysis, C.L. and L.S.; investigation, C.L., L.S. and G.Q.; data curation, C.L.; writing—original draft preparation, C.L. and L.S; writing—review and editing, C.L., L.S. and G.Q.; visualization, L.S.; supervision, G.Q.

Informed Consent Statement: All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Henrique, B.M.; Sobreiro, V.A.; Kimura, H. Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications* **2019**, *124*, 226–251.
2. Cenesizoglu, T.; Dionne, G.; Zhou, X.; others. *Asymmetric effects of the limit order book on price dynamics*; CIRRELT, Centre interuniversitaire de recherche sur les réseaux d'entreprise . . . , 2016.
3. Mondal, P.; Shit, L.; Goswami, S. Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices. *International Journal of Computer Science, Engineering and Applications* **2014**, *4*, 13.
4. Tran, D.T.; Magris, M.; Kanniainen, J.; Gabbouj, M.; Iosifidis, A. Tensor representation in high-frequency financial data for price change prediction. 2017 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2017, pp. 1–7.
5. Yun, K.K.; Yoon, S.W.; Won, D. Prediction of stock price direction using a hybrid GA-XGBoost algorithm with a three-stage feature engineering process. *Expert Systems with Applications* **2021**, *186*, 115716.
6. Kercheval, A.N.; Zhang, Y. Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance* **2015**, *15*, 1315–1329.
7. Tsantekidis, A.; Passalis, N.; Tefas, A.; Kanniainen, J.; Gabbouj, M.; Iosifidis, A. Forecasting stock prices from the limit order book using convolutional neural networks. 2017 IEEE 19th conference on business informatics (CBI). IEEE, 2017, Vol. 1, pp. 7–12.
8. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation* **1997**, *9*, 1735–1780.
9. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, *preprint*, arXiv:1406.1078.
10. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Advances in neural information processing systems (NIPS)* **2017**, *31*, arXiv:1706.03762.
11. Zhang, Z.; Zohren, S.; Roberts, S. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing* **2019**, *67*, 3001–3012.
12. Zhang, Z.; Zohren, S. Multi-horizon forecasting for limit order books: Novel deep learning approaches and hardware acceleration using intelligent processing units. *arXiv* **2021**, *preprint*, arXiv:2105.10430.
13. Baron, M.; Brogaard, J.; Hagströmer, B.; Kirilenko, A. Risk and return in high-frequency trading. *Journal of Financial and Quantitative Analysis* **2019**, *54*, 993–1024.
14. Gould, M.D.; Porter, M.A.; Williams, S.; McDonald, M.; Fenn, D.J.; Howison, S.D. Limit order books. *Quantitative Finance* **2013**, *13*, 1709–1742.
15. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, *preprint*, arXiv:1409.0473.
16. Beaton, A.E. The Use of Special Matrix Operators in Statistical Calculus. *Educational Testing Service* **1964**, *Research Bulletin*, RB–64–51.
17. Goodnight, J.H. A tutorial on the SWEEP operator. *The American Statistician* **1979**, *33*, 149–158.