Iranian Journal of Finance

Print ISSN 2676-6337
Online ISSN 2676-6345

Automation of Algorithmic Trading Strategies in Artificial Financial Markets by Combining Machine Learning Techniques and Agent-based Modeling

Mohammad Hossein Poostforoush

Ph.D. Candidate in Information Technology Management, Department of Management, Isfahan (Khorasgan) Branch, Islamic Azad University, Isfahan, Iran. (Email: m.poustfroush@gmail.com)

Amirhassan Monajemi*

*Corresponding Author, Senior Lecturer, School of Computing, National University of Singapore, 117417, Singapore. (Email: amir@comp.nus.edu.sg)

Saeed Daei-Karimzadeh

Associate Prof., Department of Economics, Isfahan (Khorasgan) Branch, Islamic Azad University, Isfahan, Iran. (Email: saeedkarimzade@yahoo.com)

Saeed Samadi

Associate Prof., Department of Economics, Faculty of Administrative Science and Economics, University of Isfahan, Isfahan, Iran. (Email: s.samadi@ase.ui.ac.ir)

Iranian Journal of Finance, 2025, Vol. 9, No.3, pp. 95-134.
Publisher: Iran Finance Association
doi: https://doi.org/ 10.30699/ijf.2025.458379.1472
Article Type: Original Article

Received: May 19, 2024
Received in revised form: March 19, 2025
Accepted: April 28, 2025
Published online: July 20, 2025

Abstract

This study aims to demonstrate the performance of algorithmic trading strategies compared to traditional trading methods in artificial financial markets. This research uses a hybrid model based on agent-based modeling and machine learning methods to simulate agents' behavior in an artificial financial market. This model includes two categories, traditional agents and intelligent agents. Traditional agents are divided into three groups: liquidity providers, liquidity consumers, and noise traders. Intelligent agents are trained using deep learning techniques and recurrent neural networks. Based on the developed algorithms, the agent-based model simulates both categories of traditional and trained agents in an artificial financial market. Sensitivity analysis tests were used to test the validity and reliability of the model, and the values of the fat-tailed distribution of returns, volatility clustering, autocorrelation of returns, long memory in order flow, concave price impact, and extreme price events are calculated in the model and compared with the standardized values. Historical data was used to predict stock prices, and model simulations were used to generate trading signals and update the limited order book. The results of executing the model show the ability of intelligent agents to trade in artificial financial markets compared to traditional agents.

Keywords: Algorithmic Trading, Machine Learning Methods, Agent-based Modeling, Recurrent Neural Networks

JEL Classification: G17, C63, C45

Introduction

This study attempts to develop a new model based on designing a framework for optimizing sales and market regulator decisions to create new insights for modern and dynamic financial markets.

The primary role of financial markets is to create the context for transactions between people who want to buy or sell the same commodity and provide a platform for the exchange of liquidity. Considering all relevant information and asset pricing, these markets act as a pricing tool. Liquidity and price formation are emerging features of the low-level interactions of buyers and sellers who make up the market.

Many efforts have been made to create a consistent and independent trading system. The inspiration for such trading systems comes from fields ranging from fundamental analysis and economic modeling to dynamic computing, machine learning, and even news mining.

The algorithmic trading strategies used today can be broadly classified into the following nine categories:

- Trend-following approaches: The most common algorithmic trading approaches follow moving average trends, channel failure, price level movements, and related technical indicators.
- Arbitrage Opportunities: Buying stocks at a lower price in one market and selling them at a higher price in another market simultaneously presents the price difference as a risk-free or arbitrage profit.
- Rebalancing in Index Stock Mutual Funds: These funds have defined re-equilibrium periods to balance their resources with their respective benchmarks. This creates profitable opportunities for algorithmic traders to invest in expected trades before the fund rebalances.
- Mathematical model-based approaches: Proven mathematical models, such as the Delta-neutral trading approach, allow trading with a combination of options and basic security. (Delta-neutral is an asset portfolio approach consisting of different situations compensating for positive and negative deltas so that the total delta of the desired assets reaches zero.)
- Scope of trades (average return): The average return approach is based on the concept that high and low prices are the temporary phenomenon assets that periodically return to their average value (average value). Identifying and defining the price range and implementing an algorithm based on it allows transactions to be done automatically if the price of the asset breaks and deviates from its defined range.
- Volume-weighted Average Price: This approach breaks down a large order into smaller parts, and markets dynamically define parts that are smaller than the order using specific stock historical volumes. The goal is to bring the order closer to the weighted average price volume.
- Time Weighted Average Price: This approach breaks a large order into smaller pieces and offers smaller, dynamically determined orders in the market using the time intervals between the start and end times. The goal is to bring the order closer to the average price between the start and end time, thereby minimizing market impact.
- Percentage of volume: This algorithm sends partial orders according to the defined participation ratio and the volume of transactions in the markets until the commercial order is filled. The "step-by-step approach" sends orders at a percentage of the user-defined market volume and increases or decreases this participation as the stock price reaches the user-defined level.

- Implementation Shortfall: This approach aims to minimize the cost of executing an order through the real-time exchange in the market, thus saving the cost of the order and providing the opportunity to execute with a delay. This stock approach increases the target participation rate if the stock price moves favorably and decreases if the stock price moves unfavorably (Johnson, 2010).

Currently, the family of financial simulations is witnessing the addition of a new member, agent-based simulations, in the market. Since the early 1990s, there has been a growing interest in using agent-based methods to gain insight into market microstructure and test sales approaches in financial communities (both industrial and academic). Research on agent-driven financial markets naturally provides a tremendous opportunity for interdisciplinary competition, as it often involves financial engineers, economists, computer scientists, mathematicians, statisticians, physicists, and others in joint projects (Guo, 2005).

The simulation process for an agent-based model involves a four-step method: first, the agent-based model is constructed, and artificial time-series data is generated; then, a deep multilayer artificial neural network is designed and trained with artificial data. In the third step, the trained neural network must be experimentally validated with real data; in the fourth step, a trained and accredited deep neural network is applied to analyze economic policy (van der Hoog, 2017).

The market orders book keeps all the sales orders that the customer enters. All features of incoming orders, including prices, number of shares or contracts, types of orders, and identification of participants, are recorded. Purchase orders (purchase offers) are arranged in descending order (from highest to lowest price). Higher-priced bids have a higher priority in terms of matching. Equal price bids are made using the FIFO algorithm (first input, first output). Sales orders (sales requests) are arranged in ascending order (from the lowest to the highest price). The order book will be updated in response to each order sent by customers. The limited order book is very similar in structure to the market order book. The only difference is that this book is made by market customers and based on market data that brokers send in response to orders sent by customers to market customers (Donadio & Ghosh, 2019).

A review of previous research shows that historical data has often been used as input to the agent-based model to simulate financial market strategies in the methods used in these studies. The use of historical data to simulate financial market strategies has the disadvantage that the simulation of agents' behavior is based on this data, and agents do not have the opportunity to identify the actual market behavior and predict events that may affect market prices at any time. Also, simulation with historical data in two-way markets, it is impossible to provide realistic offers to offer stock selling prices and offer selling prices based on the long-term trend of market price changes and filling the list of sales orders in the dynamic order book, so design a hybrid model that can make a price prediction as to the input of the agent-based model, and the agent-based model can simulate market behavior based on this input can be helpful. The development of such hybrid models can also predict future price changes and changes in other variables affecting the market situation, such as trading volume, as the leading agents in the simulation, and simulate the effects of these variables on market decisions.

Literature Review

Currently, the general and academic literature on algorithmic transactions is extensive and covers a wide range of computer transactions controlled by algorithms and other specific cases. In these transactions, computers communicate directly with the trading system and record orders without human intervention. Computers receive market information and possibly other important information very quickly, then build algorithms based on it and modify transactions in a fraction of a second. To achieve different goals, various types of algorithms are used, for example, some of them seek to identify arbitrage opportunities (which include finding minor differences in the exchange rate between currencies), others seek to maximize the execution of large orders with Minimal cost, and some are looking for a long-term trading approach to make a profit.

Machine learning models and statistical methods can be further characterized as parametric or nonparametric. Parametric models consider a limited set of parameters and try to respond to the model as a function of input variables and parameters. These models have limited flexibility due to limited parameter space and cannot use complex patterns in extensive data. As a general rule, examples of parametric models include ordinary least squares linear regression models, polynomial regression, mixture models, neural networks, and hidden Markov models. Nonparametric models consider the parameter space as an infinite dimension, equivalent to introducing a hidden function. Examples of nonparametric models include core methods such as Support Vector Machines and Gaussian Processes. Parametric or nonparametric neural networks depend entirely on how they are installed. For example, a neural network's parameter space can be considered an infinite dimension, and therefore, neural networks can be described as nonparametric (Dixon, 2020).

A considerable number of study opportunities use machine learning algorithms to create trading rules. These algorithms are so-called technical analyses that take advantage of interesting opportunities for short-term statistical attention by technical indicators such as momentum and trending. However, the absence of a solid mathematical foundation for technical analysis has meant minimal presence in the academic literature. Furthermore, during the 1960s, trading rules based on technical indicators were studied and found unprofitable (Fama & Blume, 1966; Alexander, 1961). This work led notable academics to dismiss technical analysis and support the efficient market hypothesis (Fama, 1970). However, the major problem with the early technical analysis studies was the ad hoc specifications of the trading rules that suggested data dredging (Booth, 2016).

It is easier to understand similar phenomena in agent-based models than in representations of mathematical models. This is because agent-based models, unlike mathematical equations made of mathematical symbols, are made of individual objects and simple rules for shifting their behavior. In natural discourse, experiences are usually described as the interactions of individuals, contrary to the rate of change in differential equations. Individual factors can be understood by designing mental experiences around them. Therefore, the language and concepts used in agent-based modeling are very close to our natural language and natural thinking. However, these changes in science have not led mainly to significant changes in the world of education in practice. There are many reasons for the slow pace of change in technology transfer to the education system. One of these obstacles is the lack of a broad understanding of the benefits of such a transfer (Wilensky & Rand, 2015).

The recurrent neural network is composed of several layers of nodes, in which the first layer is the input layer, and the number of nodes in the first layer corresponds to the number of input properties. The last layer in the recursive neural network is the output layer, where the number of nodes corresponds to the number of output signals. One or more hidden layers separate the input and output layers. The nodes are completely connected in coordinated layers. Each node takes information from the previous layer and transfers it to the bottom layer. The recursive neural network performs the analysis of its input variables. The node is moved to the next layer if the sum reaches the threshold. Otherwise, it remains inactive. Therefore, each network is trained when receiving some inputs. As a result, a learning process takes place. Learning in the Perceptron network is done by changing the connection weight after processing each piece of data. It is based on the number of errors compared to the predicted result (Aloud, 2020).

Kim and Markowitz proposed one of the first agent-based models of financial markets in the stock market. Kim and Markowitz sought the relationship between factors related to portfolio insurance approaches and stock market volatility through agent-based modeling; Kim and Markowitz developed the theoretical foundations for the reasons for the collapse of the US stock market in 1987; Other models were developed in later years by researchers such as Levy et al. (1994), Huang and Solomon (2001) in collaboration with some physicists, all of whom developed agent-based modeling in financial markets (Samanidou et al., 2007).

Research Methodology

1. Hybrid model structure

In this research, two categories of agents are used to perform model simulations. The first category is traditional agents who make decisions based on traditional models with conventional market approaches. These approaches are classified into three groups: liquidity consumers, providers, and noise traders. The second category of agents is the intelligent agents, which are trained by a recurrent neural network.

To create the model, the first group of agents is randomly generated with a probability of δ_n for noise traders, another group of agents with a probability of δ_c for liquidity consumers, and the third group with a probability of δ_p for liquidity providers. In each period, an agent is randomly selected. In the next step, the order's type, price, and volume are defined according to the rules of the groups. The order is then recorded in the order book, which matches the time-price pattern. If a limited order initiates no transaction, that order will remain in the order book until the book is completed.

The second category is intelligent agents who are first implemented in the NetLogo software environment with the probability $\delta_t = 1 - \delta_n + \delta_c + \delta_p$ for implementation. Then, using NetLogo software extensions, Python software, and required packages (TensorFlow and Keras), the final stock price forecast for the following market step is done using deep learning methods. The predicted final price is entered into the agent-based model as a variable and according to the rules of this agent and comparing this predicted price with the prices recorded in the book of limited orders, the trading signal (buy a stock, sell a stock, hold stock) is issued by the model and the limited order book will be updated. The conceptual model of the final hybrid model is shown in Figure 1.

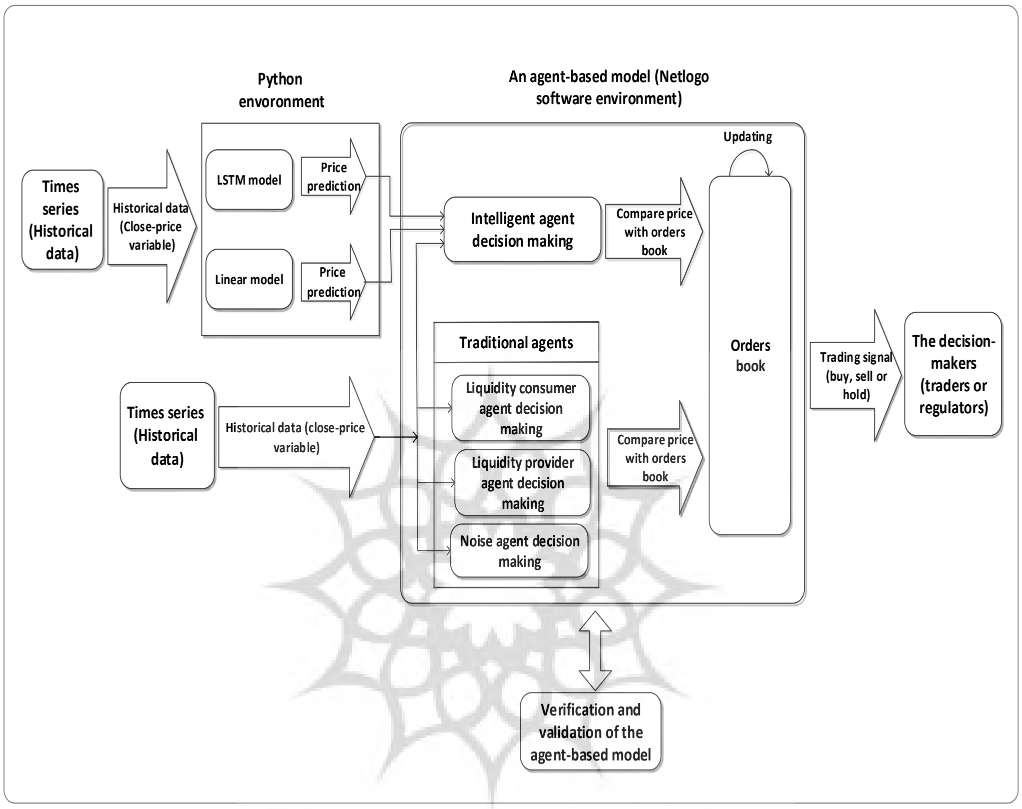
Python envoronment
An agent-based model (Netlogo software environment)
Updating
Times series (Historical data)
Historical data (Close-price variable)
LSTM model
Price prediction
Linear model
Price prediction
Intelligent agent decision making
Compare price with orders book
Orders book
Traditional agents
Liquidity consumer agent decision making
Liquidity provider agent decision making
Noise agent decision making
Times series (Historical data)
Historical data (close-price variable)
Compare price with orders book
Trading signal (buy, sell or hold)
The decision-makers (traders or regulators)
Verification and validation of the agent-based model

Figure 1. Schematic of the hybrid model

2. Traditional agents

In the first step, traditional agents generate the trading signal and update the limited order book using the Monte Carlo simulation method.

To do this, the model described by Oesch (2014) defines the types of agents in the market. This model uses a limited two-way market order book used in most modern exchange markets. The model does not allow an agent to do anything in a period, as the model is implicitly located at the time of events. Transactions occur sequentially, meaning only one agent can operate at each step. Agents can place limited orders or market orders and cancel previously registered orders. The market has three groups of agents: liquidity consumers, liquidity providers, and noise traders.

2.1. Liquidity consumers

This group is a large group of traders who make decisions based on beliefs or stock portfolios that have been rebalanced to their needs. This group includes investment institutions such as pension funds, banks, insurance companies, and other investors in fundamental financial markets. This group creates large orders that want to operate with the least impact on the market and costs. The group buys or sells large stock orders in one day to minimize the impact of price and transaction costs. The probability that this group of agents buys or sells is determined by equal probability.

The algorithm describing the logic of this group of traders is presented in Table 1 (Booth, 2016).

Table 1. Liquidity consumer algorithm

```
Pseudocode for Liquidity Consumers
1: If the start of the day, then
2:     if rand() < 0.5 then
3:         Buying
4:     else
5:         Selling
6:     end if
7:     Initial order volume, $h_0$ = $U(h_{min}, h_{max})$
8: end if
9: if rand() < $\delta_c$ then
10:     if $h_t \le \Phi_t$ then
11:         Submit market order with volume $v_t = h_t$
12:     else
13:         Submit market order with volume $v_t = \Phi_t$
14:     end if
15: end if
16: $h_t = h_t - v_t$
```

The initial volume h_o of a large order is drawn from a uniform distribution between h_{min} and h_{max}. To execute the large order, a liquidity consumer agent looks at the current volume available at the opposite best price, Φ_t. If the remaining volume of his large order, h_t, is less than Φ_t, the agent sets this period's volume to $v_t = h_t$. Otherwise, he takes all available volume at the best price, $v_t = \Phi_t$. For simplicity, liquidity consumers only utilize market orders.

2.2. Liquidity providers

This group of market participants tries to profit from the difference between the buy and sell price through the supply of liquidity on both sides of the bid price and the asking price from the order book, and maintains an almost neutral position during the trading day. Market makers play this role in traditional markets, but in modern electronics markets, hypothetically, any agent can follow such an approach. Table 2 describes the trading logic of this group of traders (Booth, 2016):

Table 2. Liquidity provider algorithm

```
Pseudocode for Liquidity Providers
1: if rand() < $\delta_p$ then
2:     Cancel any existing orders
3:     if predicts next order is buy then
4:         Submit sell at best price with volume = $U(v_{min}, v_{max})$
5:         Submit buy at best price with volume = $v^-$
6:     else
7:         Submit buy at best price with volume = $U(v_{min}, v_{max})$
8:         Submit sell at best price with volume = $v^-$
9:     end if
10: end if
11: Update buy/sell prediction with w-period rolling mean
```

Each liquidity provider always has a limit order on both sides of the order book (buy and sell). If either of the limit orders is executed, it will be replaced by a new one the next time the liquidity provider gets to trade. When a liquidity provider turns to trade, he will update his prediction for the next period's order sign and change his order volumes. If the predictor, for example, forecasts a buy order, the liquidity provider will adjust his current ask and bid volume by setting the sell limit order volume v^+ (the order which is expected to be executed next) to a uniformly distributed random variable between v_{min} and v_{max}, and the unexpected buy limit order's volume to v^-.

This asymmetric exposure strategy can be explained by the liquidity provider's task to keep the market efficient or by the argument that the liquidity provider has a budget restriction. If a budget restriction is present, he will always want to allocate his resources to the position where he most likely can trade soon to earn the bid-ask spread. However, this is only speculation and an open question for further investigation. A simple rolling-mean estimate with ω periods is chosen to predict the sign of the following order. While this linear predictor might not be optimal, it has been used to stick with it for simplicity.

2.3. Noise traders

The noise traders represent all other strategies on the market and can be viewed as speculative traders, and can be fitted to have empirical order probabilities. The noise trader algorithm is described in Table 3 (Booth, 2016).

Table 3. Noise trader algorithm

Pseudocode for Noise Traders

```
1: if rand() < $\delta_n$ then
2:     if rand() < 0.5 then
3:         Selling
4:     else
5:         Buying
6:     end if
7:     Generate U(0 , 1) to determine action, $\lambda_m$ , $\lambda_l$, and $\lambda_c$
8:     switch action do
9:         case Submit Market Order
10:            Submit market order with volume calculated by Equation $v_t = exp(\mu + \sigma u_v)$
11:        case Submit Limit Order
12:            Generate U(0 , 1) for action, $\lambda_{crs}$ , $\lambda_{insp}$ , $\lambda_{spr}$ & $\lambda_{offspr}$
13:            switch Limit Order do
14:                case Crossing limit order
15:                    Submit a limit order at the opposing best price using Equation $v_t = exp(\mu + \sigma u_v)$
16:                case Inside spread limit order
17:                    Generate a random value, $p_{insprd}$ = U(BestBid , BestAsk)
18:                    Submit limit order at price $p_{insprd}$ using Equation $v_t = exp(\mu + \sigma u_v)$
19:                case Spread limit order
20:                    Submit a limit order at the best price using Equation $v_t = exp(\mu + \sigma u_v)$
21:                case Off-spread limit order
22:                    Generate a random value, $rp_{offsprd}$ using Equation $xmin_{offspr} * (1 - u_o)^{\frac{-1}{\beta-1}}$
23:                    Submit order at a price $rp_{offsprd}$ outside of the spread using Equation $v_t = exp(\mu + \sigma u_v)$
24:        case Cancel Limit Order
25:            Cancel the oldest order previously submitted
26: end if
```

The noise agents randomly decide on whether to buy or sell in each period with equal probability. Once decided, they randomly choose to place a market order, a limit order, or to cancel an existing order. If a limit order is chosen, they face four further options. With probability λ_{crs}, they cross the spread and place the limit order at the opposing best, making the order an effective market order. The order will remain in the order book if it is not completely filled. With probability λ_{insp}, they decide to place the order between the bid and ask spread, and with probability λ_{spr}, they place the limit order at the best price available. Finally, with probability λ_{offspr}, the noise traders decide to place their limit order in the book. The relative off-spread limit order price is distributed with a power-law as:

$$xmin_{\text{offspr}} * (1 - u_o)^{\frac{-1}{\beta-1}} \qquad (1)$$

Where u_o is a uniformly distributed random variable between 0 and 1. Most orders fall into this last class of limit orders. The volume for each order is drawn from a log-normal distribution:

$$v_t = exp(\mu + \sigma u_v) \qquad (2)$$

Where u_v again is a uniformly distributed random variable between 0 and 1, and μ and σ are parameters, which can be fit empirically.

The sum of these probabilities must equal one ($\lambda_{crs} + \lambda_{insp} + \lambda_{spr} + \lambda_{offspr} = 1$). To prevent spurious price processes, noise traders' market orders are limited in volume, so they cannot consume more than half of the total available volume of the opposing side. Another restriction is that noise traders will ensure that no side of the order book is empty and place limit orders appropriately.

3. Intelligent agents

For intelligent agents, stock price prediction uses machine learning methods and a recurrent neural network using historical time series data.

3.1. Machine learning methods

Machine learning addresses a fundamental prediction problem: Construct a nonlinear predictor, $\hat{Y}(X)$, of output, Y, given a high-dimensional input matrix $X = (X^{(1)}, \dots, X^{(p)})$ of p variables. Machine learning can be viewed as studying and constructing an input-output map of the form Y = F (X), where $X = (X^{(1)}, \dots, X^{(p)})$. The output variable, Y, can be continuous, discrete, or combined. For example, in a classification problem, $F: X \rightarrow Y$ where $Y \in \{1, \dots, k\}$ and k is the number of categories. When Y is a continuous vector and f is a semi-affine function, then the linear model is recovered as:

$$Y = AX + b \qquad (3)$$

3.2. Recurrent neural network

The recurrent neural network processes a series of time steps, step by step, and maintains an internal state from one time step to another. These networks are suitable for predicting and classifying sequences and are widely used to predict one-variable financial time series. These networks are learning sequences that have been successful in applications such as natural language processing, language generation, image processing, and many other tasks.

A simple RNN is formed by a repeated application of a function F_h to the input sequence $x_t = (X_1, \dots, X_T)$. For each time step t = 1,…, T, the function generates a hidden state h_t from the current input X_t and the previous output h_{t-1}:

$$h_t = F_h(X_t, h_{t1}) = \boldsymbol{\sigma}(W_h X_t + U_h h_{t1} + b_h) \tag{4}$$

For some nonlinear activation function σ(x). This simple RNN unfolds a single hidden-layer neural network over all time steps. When the output is continuous, the model output from the final hidden state, Y $= F_y(h_t)$, is given by the Semi-affine function:

$$\mathrm{Y} = F_y(h_t) = W_y(h_t) + b_y \tag{5}$$

Moreover, when the output is categorical, it is given by:

$$\mathrm{Y} = F_y(h_t) = softmax(F_y(h_t)) \tag{6}$$

Where Y has a 'one-hot' encoding - a K-vector of zeros with 1 at a single position $W = (W_h, U_h, W_y)$ and $b = (b_h,\ b_y)$ are weight matrices and offsets, respectively. $W_h \in \mathbb{R}^{H \times P}$ denotes the weights of non-recurrent connections between the input X_t and the H hidden units. The weights of the recurrence connections between the hidden units are denoted by the recurrent weight matrix $U_h \in \mathbb{R}^{H \times H}$ Without such a matrix, the architecture is simply an unfolded single-layer feed-forward network without memory, and each observation X_t is treated as an independent observation.

W_y denotes the weights tied to the output of the hidden units at the last time step, h_t, and the output layer. If the output variable is a continuous vector, $W_y \in \mathbb{R}^M$ then $W_y \in \mathbb{R}^{M \times H}$ If the output is categorical, with K states, then $W_y \in \mathbb{R}^{K \times H}$ (Dixon, 2017).

The structure of the recursive neural network is shown in Figure 2.

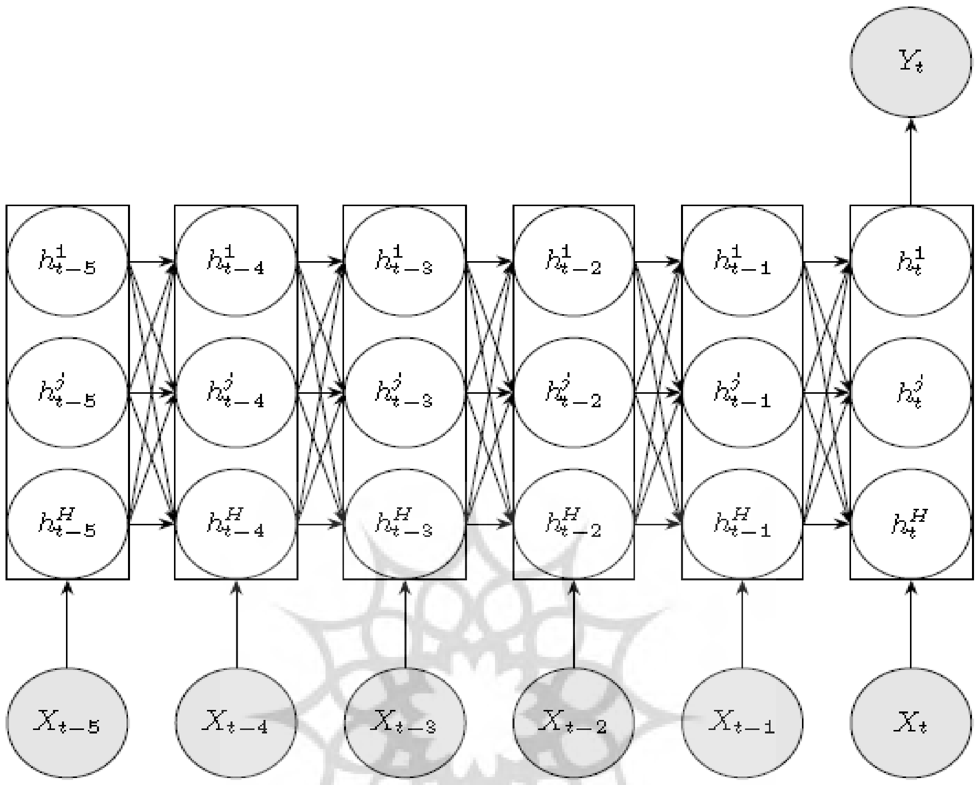

Figure 2. Recurrent neural network structure

3.2.1. Training, Validation, and Testing

To construct and evaluate a learning machine, we start by the controlled splitting of the data into training, validation, and test sets. The training data consists of input-output pairs $D = \{ Y_t, X_t\}_{t=1-(T-1)}^{N}$ .We then sequence the data to give $\mathrm{D}_{seq} = \{ Y_t, X_t\}_{t=1}^{N}$.

The goal is to find the machine sequence learner $\mathrm{Y} = F(x)$, where we have a loss function $\mathcal{L}(Y, \widehat{Y})$ for a predictor, $\hat{Y}$, of the output signal, Y. In many cases, there is an underlying probability model, $p(Y|\widehat{Y})$, then the loss function is the negative log probability $\mathcal{L}\big(Y, \widehat{Y}\big) = -\log p(Y|\widehat{Y})$. For example, under a Gaussian model, $\mathcal{L}\big(Y, \widehat{Y}\big) = ||Y - \hat{Y}||^2$ is an L^2 norm, for binary classification, $\mathcal{L}\big(Y, \widehat{Y}\big) = -Y \log \hat{Y}$ is the negative cross-entropy. In its simplest form, we then solve an optimization problem:

$$\underset{W,b}{minimize}\, f(W, b) + \lambda \phi(W, b) \tag{7}$$

$$f(W,b) = \frac{1}{N}\sum_{t=1}^{N} \mathcal{L}\left(Y_t, \hat{Y}(x_t)\right) \tag{8}$$

with a regularization penalty, $\phi(W,b)$.

Here $\boldsymbol{\lambda}$ is a global regularization parameter that we tune using the model's out-of-sample predictive mean-squared error (MSE) on the verification data. The regularization penalty, $\phi(W,b)$, introduces a Bias-variance tradeoff. $\nabla\mathcal{L}$ is given in the closed form by a chain rule and, through back-propagation on the unfolded network, the weight matrices $\widehat{W}$ are fitted with stochastic gradient descent.

3.2.2. Predictor Selection and Dropout

Dropout is a model or variable selection technique. Input space X needs dimension reduction techniques designed to avoid over-fitting during the training process. Dropout removes input variables in X_t randomly with a given probability θ. The probability, θ, can be viewed as a further hyperparameter (like $\boldsymbol{\lambda}$) tuned via cross-validation. Heuristically, if there are P=100 variables in X_t, then a choice of θ=0.1 will result in a search for models with 10 variables. The dropout architecture with the stochastic search for the predictors can be used.

$$D_i \sim Ber(\theta) \tag{9}$$

$$\widetilde{X_t} = D \star X_t, t = 1, \dots, T \tag{10}$$

$$h_t = F_h(W_h\widetilde{X_t} + U_h h_{t-1} + b_h) \tag{11}$$

Effectively, this replaces the input X_t by $D \star X_t$, where $\star$ denotes the element-wise product and D is a 'dropout operator' - a vector of independent Bernoulli, $Ber(\theta)$, distributed random variables. The overall objective function is closely related to ridge regression with a g-prior (Heaton et al., 2017). Note that dropout is not applied to the recurrent connections, only the non-recurrent connections.

Graves (2013) provided evidence of the success of RNNs by applying dropout only to the non-recurrent connections in an LSTM (M. Dixon, 2017).

3.3. LSTM neural network

The Long Short-Term Memory (LSTM) unit was introduced by Hochreiter and Schmidt Huber in 1997.

LSTM is a recurrent neural network architecture designed to store and access information better than the traditional version. A traditional recursive neural network (if large enough) should theoretically be able to generate sequences of any complexity. However, it cannot store information related to past inputs for a long time (Hochreiter et al., 2001).

This feature weakens the network's ability to model long-term structures, and this "forget" causes these types of networks to be unstable during sequence generation. The problem (common to all conditional production models, of course) is that if the network predictions depend only on a few recent inputs and these inputs are generated by the network itself, there is very little chance of correcting past errors by the network.

Having a longer-term memory stabilizes because even if the network fails to understand its recent history, it can still complete its prediction by looking back. The instability problem is especially acute when dealing with decimal data, as forecasts can distance themselves from the manifold on which the training data is placed. One solution proposed for conditional models is to inject noise into the predictions made by the network before feeding them to the next time step (Taylor & Hinton, 2009).

In the LSTM neural network, we encounter new concepts that did not exist in the traditional recursive neural network. In this network, there are three gateways through which the network controls its data flow. These three gates are: forget gate, update gate (also known as the input gate), and output gate.

The gate of forgetting is used to forget unnecessary information from the past. This gate controls the flow of information from the previous step. It determines whether the memory information is used from the previous step or not, and if something should be entered from the previous step.

The update gateway checks whether the information obtained from the current moment (t moment) is worth storing in long-term memory. This gateway determines whether new information should be used at the current time step and, if so, what rate should be used.

The output gate specifies how much information from the previous step is transferred to the next step, along with the current time step information. This gateway is used to prevent all information in C_t from being transferred to the output h_t and to transfer some of the required information to the output. The output of the gates is always between 0 and 1 and is always multiplied element by element with another input; each gate has two inputs, which are x_t and h_{t-1}. These two inputs are multiplied in two layers, gathered together, and passed through the sigmoid function. The information is always placed between intervals -1 and +1 using a hyperbolic tangent function. In addition to these three gates, there is a memory cell, which is abbreviated as C. These are new concepts in this network, and in addition to these four new concepts, the network also has a cache input (h) and an input (X) and produces two outputs (one output is C_t and the other output is h_t, which itself is divided into two parts, one part is transferred to the next time step, and the other part is used in case of need to produce output in the current time step.) (Graves, 2013).

Figure 3 shows the structure of an LSTM cell:

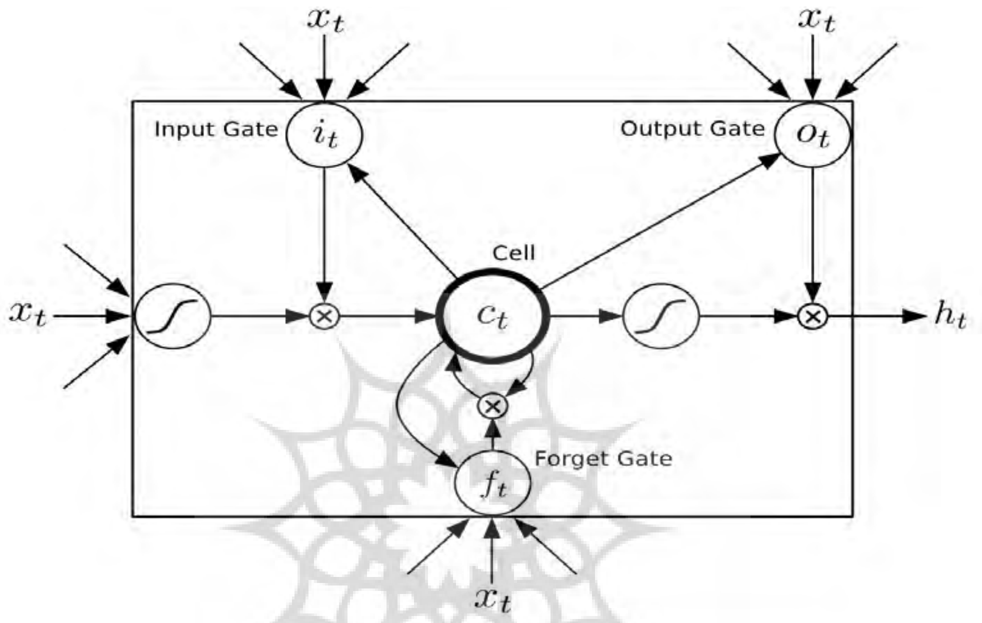

Figure 3. LSTM cell structure

Unlike a traditional recurrent neural network that merely computes a balanced sum of input signals and then passes through an activation function, each LSTM uses a C_t memory at time t. The output of h_t, or activation of the LSTM unit, is:

$$h_t = \Gamma_o . \tanh(C_t) \tag{12}$$

Where Γ_o is the output gateway that controls the amount of content delivered through memory. The output gate is calculated using the expression:

$$\Gamma_o = \sigma(W_o . [h_{t-1}, X_t] + b_o) \tag{13}$$

In which $\boldsymbol{\sigma}$ is the Sigmoid activation function. W_o is also a diagonal matrix. C_t the memory cell also with relative forgetting of current memory and adding new memory content as $\widehat{C_t}$ will be updated by:

$$C_t = \Gamma_f . C_{t-1} + \Gamma_u . \widehat{C_t} \tag{14}$$

Where the new memory content is calculated through the following equation:

$$\widehat{C}_t = \tanh(W_C \,.\, [h_{t-1}\,, X_t\,] + \, b_c) \tag{15}$$

The amount of current memory to be forgotten is controlled by the Γ_f forget gate, and the amount of new memory content to be added to the memory cell is controlled by the update gate. This is done by calculating equation (14) and the following equation:

$$\Gamma_f = \sigma(W_f \,.\, [h_{t-1}\,, X_t\,] + \, b_f) \tag{16}$$

The general structure of the LSTM recurrent neural network is shown in Figure 4.

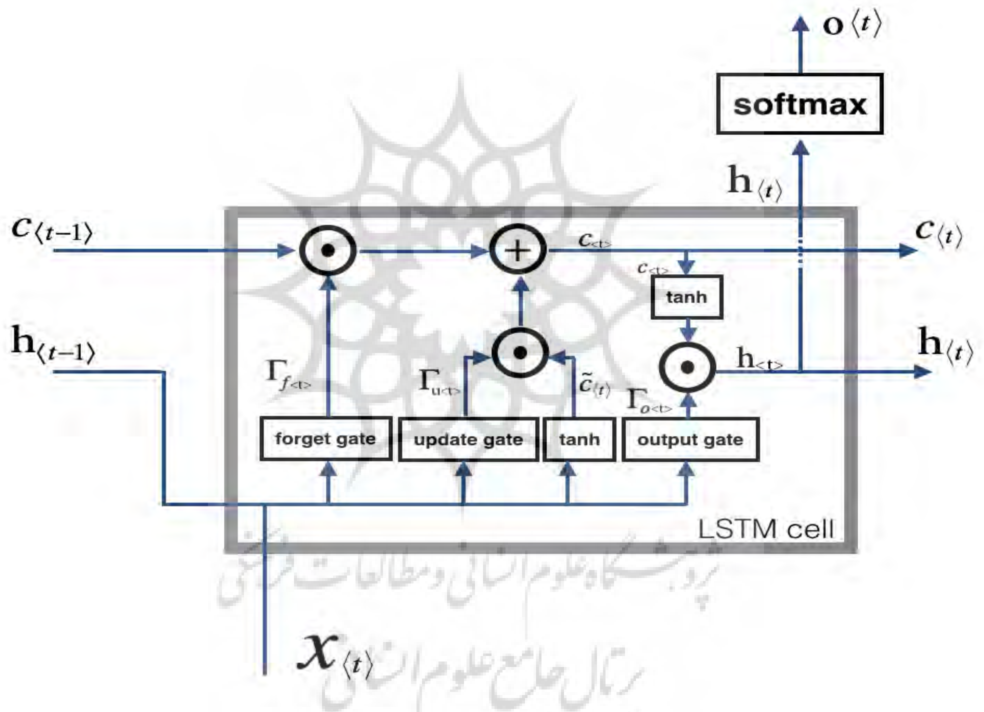

Figure 4. LSTM recurrent neural network structure

3.4. Intelligent agent algorithm

For intelligent agents, in addition to the price predictor variable for the next time, another variable is defined as the utility score of the investment.

The utility score of the investment variable represents an important meaning: agents like expected returns. At the same time, they do not like fluctuations at different levels (depending on the risk aversion of that factor).

The history of rates of return on various asset classes and elaborate empirical studies leave no doubt that risky assets command a risk premium in the marketplace. This implies that most investors are risk-averse. A prospect that has a zero-risk premium is called a fair game. Risk-averse investors consider only risk-free or speculative prospects with positive risk premiums. The greater the risk, the larger the penalty, and most investors accept this view from simple introspection. To formalize this notion, we will assume that each investor can assign a welfare, or utility, score to competing investment portfolios based on the expected return and risk. The utility score value may be viewed as a means of ranking portfolios. Portfolios receive higher utility scores for higher expected returns and lower scores for higher volatility. One reasonable function that financial theorists commonly employ is determined as follows:

$$U = E(r) - (\frac{1}{2} * A * \sigma^2) \tag{17}$$

Where U is the utility value, E(r) is the expected return, σ^2 is the variance of returns, and A is an index of the investor's aversion to taking on risk on a scale from 1 (lowest risk aversion) to 5 (highest risk aversion). The factor of $\frac{1}{2}$ is a scaling convention that will simplify calculations in later chapters. It has no economic significance, and we could eliminate it simply by defining a "new" A with half the value of the A used here (Bodie et al., 2018).

The trained agent's decisions are made based on the difference between the predicted and current prices, as well as the higher or lower decision value variable, which is taken from a utility score value. Suppose the difference between the predicted price and the current price is greater than zero (the predicted price for the next period is greater than the current price), and the value of the decision variable is greater than the utility score value. In that case, the decision to buy is made. Suppose the difference between the predicted price and the current price is less than zero (the predicted price for the next period is less than the current price), and the value of the decision variable is greater than the utility score value. In that case, the decision to sell is made. Suppose the difference between the predicted and current prices is equal to zero, or the value of the decision variable is less than the utility score value. In that case, the decision is made to keep the share or go through the buying and selling process. The intelligent agent group algorithm is given in Table 4.

Table 4. Intelligent agents' algorithm

Pseudocode for Trained Traders

```
1: get Expected_return , Risk_aversion , Volatility_of_security_returns,
        from the user interface
2: set Utility_score = Expected_return – (0.5 * Risk_aversion
        * (Volatility_of_security_returns) ^ 2)
3: get InitialPrice from the user interface
4: get PredictPrice from the PredictPrice function
5: set diff = PredictPrice – InitialPrice
6: if diff > 0 then
7:      set Decision_value  = 100 * (diff / InitialPrice)
8:      if Decision_value > Utility_score then
9:          Buy
10:     end if
11: end if
12: if diff < 0 then
13:     set Decision_value  = 100 * ((- diff ) / InitialPrice)
14:     if Decision_value > Utility_score then
15:         Sell
16:     end if
17: end if
```

4. Validation of the agent-based model

Sensitivity analysis tests are used to test the validity and reliability of the model. Then the values of the Fat-Tailed distribution of returns, Volatility clustering, autocorrelation of returns, long memory in order flow, Concave Price Impact, and Extreme price events in the model are calculated and compared with the standardized values.

4.1. Sensitivity analysis

In variance-based global sensitivity analysis, the inputs to an agent-based model are treated as random variables with probability density functions representing their associated uncertainty. The impact of the set of input variables on a model's output measures may be independent or cooperative. So the output f(x) may be expressed as a finite hierarchical cooperative function expansion using an analysis of variance (ANOVA). Thus, the mapping between input variables $x_1,\dots,x_n$ and output variables f(x)=f($x_1,\dots,x_n$) may be expressed in the following functional form:

$$f(x) = f_o + \sum_i f_i\,(x_i) + \sum_{i<j} f_{i,j}\,(x_i, x_j) + \cdots + f_{1,2,\dots,n}(x_1, x_2, \dots, x_n) \qquad (18)$$

Where f_o is the zeroth-order mean effect, $f_i(x_i)$ is a first-order term that describes the effect of variable x_i on the output f(x), and $f_{i,j}(x_i, x_j)$ is a second-order term that describes the cooperative impact of variables x_i and x_j on the output. The final term, $f_{1,2,\dots,n}(x_1,\dots,x_n)$, describes the input variables' residual n^{th} order cooperative effect. Consequently, the total variance is calculated as follows:

$$D = \int (f(x) - f_o)^2 \rho\,(x) d(x) \tag{19}$$

Where ρ(X) is the probability distribution over input variables. Partial variances are then defined as:

$$D_{i_1,\dots,i_s} = \int f^2_{i_1,\dots,i_s}(x_{i_1},\dots,x_{i_s})\rho(x)dx \tag{20}$$

Now, the total partial variance D_i^{tot} for each parameter x_i, i = $\overline{\overline{1,n,}}$ is computed as

$$D_i^{tot} = \textstyle\sum_{<i>} D_{i_1,\dots,i_s}; 1 \leq s \leq n \tag{21}$$

Where <i> refers to the summation of D that contains i. Once the above is computed, the total sensitivity indices can be calculated as:

$$S_i^{tot} = \frac{D_i^{tot}}{D}\ ; 0 \leq S_i^{tot} \leq 1 \tag{22}$$

It follows that the total partial variance for each parameter x_i is:

$$D_i^{tot} = \mathrm{D} - \mathrm{Var}(\mathbb{E}(\mathrm{f}|x_{-i})) \equiv \mathbb{E}(\mathrm{Var}(\mathrm{f}|x_{-i})) \tag{23}$$

This is a direct estimation of global sensitivity indices using values of f(x) only and a Monte Carlo algorithm (Sobol, 2001).

4.2. Fat-Tailed Distribution of Returns

Across all time scales, distributions of price returns have been found to have positive kurtosis, that is to say, they are "fat-tailed". Understanding positively kurtotic distributions is paramount for trading and risk management, as large price movements are more likely than in commonly assumed normal distributions. Fat-tails have been observed in the returns distribution of many markets. In this model, only substantial cancellations, orders that fall inside the spread, and large orders that cross the spread can alter the mid-price. This generates many periods with returns of 0, significantly reducing the variance estimate and generating a leptokurtic distribution in the short run (McGroarty et al., 2019).

4.3. Volatility Clustering

Volatility clustering refers to the long memory of absolute or square mid-price returns, meaning that significant price changes tend to follow other significant price changes. Let $X = X_{t1}, X_{t2}, \dots, X_{tk}$ denotes a real-valued, wide-sense stationary time series. Then, we can characterize long memory using the diffusion properties of the integrated series Y:

$$Y(l) = \sum_{i=1}^{l} X(t_i) \quad (24)$$

Furthermore, let:

$$Y(l) = Var(Y(t_i + 1), Y(t_i + 2), \dots, Y(t_i + l) \quad (25)$$

For some $i \in \{0,1, \dots., l\}$st. Given this, in the limit $l \to \infty$ if X is a short-memory process, then V(l) scales as O(l), whereas if X is a long-memory process, then V(l) scales as $O(l^{2H})$, for some $H \in (0.5,1)$.

The Hurst exponent, H, is defined in terms of the asymptotic behavior of the rescaled range as a function of the period of a time series as follows:

$$\mathbb{E}\left[\frac{R(n)}{S(n)}\right] = Cn^H \qquad \text{as} \quad n \to \infty \quad (26)$$

Where $\mathbb{E}[x]$ is the expected value, R(n) is the range of the first n cumulative deviations from the mean, S(n) is the series (sum) of the first n standard deviations, n is the period of the observation (number of data points in a time series), and C is a constant. The Hurst exponent describes the self-similarity of a market. Self-similarity describes how similar past market snippets are to current ones. A Hurst exponent of 0.5 means that the market follows a random walk over the long term. In this case, in the long run, any trading strategy would be a zero-sum game (excluding commissions). If the Hurst exponent exceeds 0.5, the market shows a trending behavior. Past moves are similar to current moves. Markets with a high Hurst exponent are perfect for trend following strategies. If the market went up in the past, there would be a better than 50% chance it would increase. If the Hurst Exponent is below 0.5, the market shows a reverting behavior. If it went down in history, it could reverse its direction in the future. These markets are markets for mean-reverting strategies and short-term reversal pattern analysis.

In the empirical research studies, the Hurst exponent's values vary from $H \approx 0.58$ to $H \approx 0.815$ (Lillo & Farmer, 2004).

4.4. Autocorrelation of Returns

In several markets, returns series lack significant autocorrelation, except for slightly negative autocorrelation on very short time scales. The lack of strong autocorrelation is since if returns were correlated, traders would use simple strategies to exploit the autocorrelation and generate profit. Such actions would, in turn, reduce the autocorrelation so that it would no longer remain. Evidence suggests that the mild negative correlation found on short time scales has disappeared more quickly in recent years, perhaps an artifact of the new financial ecosystem (McGroarty et al., 2019).

4.5. Long Memory in Order Flow

The probability of observing a given type of order in the future positively correlates with its empirical frequency in the past. The mean first lag autocorrelation term of the order-sign series is calculated for the model and compared with the mean Hurst's exponent of the order-sign time series (McGroarty et al., 2019).

4.6. Concave Price Impact

Understanding price impact presents one of the most dominant questions of market microstructure analysis, i.e., how trading activity leads to price changes. The early market microstructure literature describes this concept with a focus on specialist markets. In such markets, prices are quoted by a centralized market maker who receives orders from brokers and updates her quoted prices according to the incoming order flow she witnesses. From the broker's viewpoint, the price impact of his orders is a cost paid to the market maker for her continued obligation to accept his orders, i.e., a cost for immediacy. From the viewpoint of the market-maker, some information about the future prices of assets is inferred from the order flow of the brokers. This information is then captured in the market maker's quotes, reflected by the permanent price impact. The difference between the price that an order obtains and the best prevailing quote is termed the immediate price impact and is an increasing function of order size. The temporary price impact is then defined as the difference between an order's immediate and permanent impact. The empirical research shows that the impact price follows a concave volume function. Those are the impacts of price increases, which occur more quickly with changes at small and less quickly at larger volumes. The price impact follows a power-law distribution of the following form:

$$\Delta p = \frac{\eta v^{\beta}}{\lambda} \tag{27}$$

Where Δp is the change in the mid-price caused by a trader's action, v is the trade volume, η takes the value -1 in the event of a sell and +1 in the event of a buy, and λ allows for adjustment for market capitalization. Lillo et al. (2003) found the exponent β to be approximately 0.5 for small and 0.2 for large volumes. After normalizing for daily volumes, λ was found to vary significantly across stocks with a clear dependence on market capitalization M, approximated by $M \sim \lambda^{\delta}$, within the region of 0.4.

4.7. Extreme Price Events

Since the introduction of automated and algorithmic trading, recurring periods of high volatility and extreme stock price behavior have plagued the markets. Extreme Price Events are defined as an occurrence of a stock price ticking down [up] at least ten times before ticking up [down] and with a price change exceeding 0.8% of the initial price (Johnson et al., 2013).

The model

1. Development of an agent-based model

Netlogo software has been used to develop the agent-based model. The Monte Carlo simulation method has been used to simulate the behavior of three groups of liquidity consumers, liquidity providers, noise traders in the traditional agent category, and intelligent agents. The model generates trading signals (buying, selling, holding) and updates the limit order book. Since it is impossible to simulate two worlds simultaneously in the software environment of NetLogo, it is necessary to use the features of another software to train the intelligent agent. Python software was used to do this. The Python extension in the NetLogo software environment connects the two software environments. This extension allows you to call Python software and use code written in Python in the NetLogo software environment. There are several ways to install and use the programming environment and the required Python packages. One of the best ways to do this is to install Anaconda software. Then we need to install the required version of Python, TensorFlow, Cross packages, and other required packages in the Anaconda software environment. TensorFlow is an open-source library for developing and teaching machine learning models developed by Google. TensorFlow can be thought of as an infrastructure layer for distinctive programming. The TensorFlow version is selected according to the available hardware features. Cross is a high-level software interface that solves machine learning problems, focusing on deep learning. This tool provides the basic elements for developing high-repetition machine learning solutions. Two main packages, TensorFlow and Cross, installed in the Python environment, have been used to perform price prediction in the Python software environment.

2. Predicting Price for Intelligent Agents

In this research, two linear and LSTM models have been used to predict stock prices. In the user interface of NetLogo software, the option to select either of these two models is embedded for users. If one selects any options, the functions required to predict the price are called and executed from the Python software environment. Using TensorFlow and Cross tools, intelligent agents are trained using historical time series data, and price prediction is done. This price prediction is used by intelligent agents in the NetLogo software environment as a variable to decide on the trading signal.

2.1. Price Predicting Using the Linear Model

The simplest way to predict a model is to place a linear conversion between the input and output of the model, in which case the output of a time step depends only on that step. Using this method, the model makes independent predictions in Consecutive time steps, and there is no interaction between predictions at each period. The linear model diagram is shown in Figure 5.

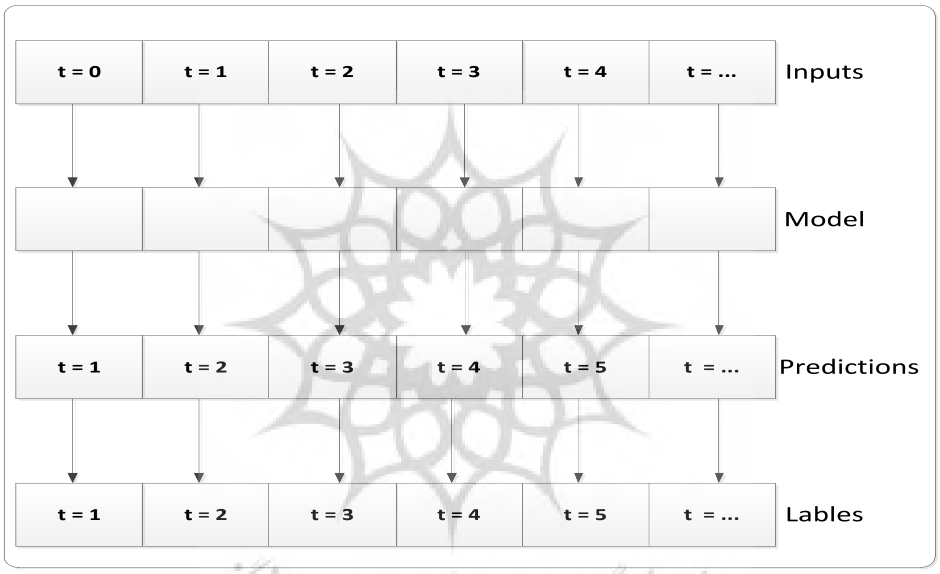

Figure 5. Linear model

2.2. Price Predicting Using the LSTM Model

In this model, the information from 22 days ago was used as the input layer. The reason for choosing 22 is that, other than weekends and public holidays, the financial markets have approximately 22 working days per month. However, this choice is not a limitation in the model, and by changing the relevant variable, this parameter can be changed in the model. Using this network, stock price prediction for the next day of the market is done using the information of the previous 22 working days. The model diagram is shown in Figure 6.

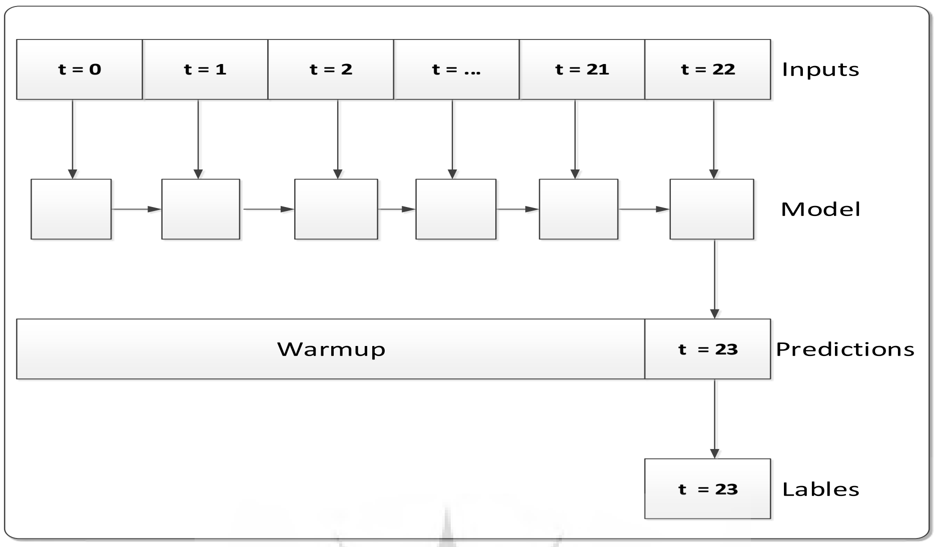

Figure 6. Recurrent neural network (LSTM) model

Results

1. Initialize free parameters of the model

An extensive grid search of the input space was performed to find the set of parameters that produce results most similar to those reported in the literature, and to further explore the influence of input parameters. This led to the "optimal" set of parameters in Tables 5 and 6.

The set of general parameters listed in Table 5 has only minor influences on the model results. On the other hand, the trading probabilities of the different agent groups have a strong influence on the price series and its return statistics, but less influence on the market impact function and the statistical efficiency of the market. The most important parameters for the form of the market impact regarding order sizes in terms of the temporal behavior of the function are the liquidity consumer and liquidity provider parameters.

Table 5. Standard setting for free parameters

Market parameters	**Setting**
Initial Price	100
Initial Spread	0.05
Tick Size	0.01
Agent group	**Action probability**
δ_c	0.10
δ_p	0.15
δ_n	0.55
δ_t	0.20
Liquidity Consumer Parameters	**Setting**
h_{min}	1
h_{max}	100000
Liquidity Provider parameters	**Setting**
v_{min}	1
v_{max}	200000
v^{-}	1
ω	50

Standard settings for noise traders' group parameters are listed in Table 6. These parameters are needed to set up the noise traders.

Table 6. Noise trader settings

Order direction	Probability
buy or sell	0.5
Event Type	Probability
submit a market order	$\lambda_m = 0.03$
submit a limit order	$\lambda_l = 0.54$
cancel a limit order	$\lambda_c = 0.43$
Limit Order Type	Probability
crossing the limit order	$\lambda_{crs} = 0.0032$
inside-spread limit order	$\lambda_{inspr} = 0.0978$
spread limit order	$\lambda_{spr} = 0.1726$
off-spread limit order	$\lambda_{offspr} = 0.7264$
Order Size Type	Parameters of Log-normal Distribution
market order size	$\mu_{mo} = 7 \quad \sigma_{mo} = 0.1$
limit order size	$\mu_{lo} = 8 \quad \sigma_{lo} = 0.7$
Limit Price Type	Parameters of Power-law Distribution
off-spread relative price	$xmin_{offspr} = 0.05 \quad \beta_{offspr} = 2.72$

If the model's free parameters are chosen too far from the default values, much larger jumps and long periods where the price does not change can result.

2. Model calibration

Calibration determines parameters so that model and market prices match closely for a given set of liquidly traded instruments (Hirsa & Neftci, 2013). Calibration involves adjusting the model parameters so that the model's outputs are consistent with observed market prices or returns (Crooks et al., 2018).

These techniques can be used to fine-tune the ABM model to reflect real-world scenarios better and improve the model's predictive capabilities:

- Sensitivity analysis: Evaluating how changes in model parameters affect the model output. Sensitivity analysis helps identify which parameters have the most significant impact and should be prioritized during calibration (Crooks et al., 2018) .
- Parameter sweeping: Systematically varying one or more parameters across their possible range and running multiple simulations. This helps identify the combination of parameters that best fit the observed data (Crooks et al., 2018).
- Optimization algorithms: Using algorithms such as Genetic Algorithms, Simulated Annealing, or Particle Swarm Optimization to automatically search for the best parameter values that minimize the difference between the model output and the real-world data (Calvez & Hutzler, 2005).
- Bayesian calibration: Incorporating prior knowledge about parameters and updating them based on observed data using Bayesian methods. This approach provides a probabilistic framework for parameter estimation and quantifies uncertainty (Frazier, 2018).
- Data Assimilation: Integrating real-time data into the model to dynamically adjust parameters. Techniques such as the Ensemble Kalman Filter can be used for this (Evensen, 2003).
- Machine Learning: Using machine learning techniques to build surrogate models that approximate the behavior of the ABM. These surrogate models can quickly explore the parameter space and identify optimal values (Calvez & Hutzler, 2005).

This paper uses sensitivity analysis and parameter sweep methods to calibrate the model.

The results of the sensitivity analysis test (Figure 7) show that the initial values selected for the model (Tables 5 and 6) can effectively reduce the uncertainties in the parameter space. The output parameters are shown in Table 8.

The model's input parameters were set to the values in Table 7 for parameter sweeping. Then, 10,000 samples were taken from the parameter space with the input parameters uniformly distributed in the ranges. After 100,000 runs, the posterior model was obtained.

3. Implementation of the sensitivity analysis test

In this model, 20 input parameters and four output parameters are considered. The parameters related to the groups of traders and the range of initialization are given in Table 7.

Table 7. Input parameter ranges for global sensitivity analysis

Parameter	Symbol	Setting
Probability of Liquidity Providers acting	δ_p	[0.05, 0.95]
Probability of Liquidity Consumers acting	δ_c	[0.05, 0.95]
Probability of Noise Traders acting	δ_n	[0.05, 0.95]
Probability of Trained Traders acting	δ_t	[0.05, 0.95]
Liquidity Providers parameters		
Max order volume	v_{max}	$[\mathbf{10^3}, \mathbf{10^6}]$
Rolling mean period	ω	$[\mathbf{10}, \mathbf{10^3}]$
Liquidity Consumers parameters		
Max order volume	h_{max}	$[\mathbf{10^3}, \mathbf{10^6}]$
Noise Traders parameters		
Market order probability	λ_m	[0, 1]
Limit order probability	λ_l	[0, 1]
Cancel order probability	λ_c	[0, 1]
Market order size	μ_{mo}	[2, 10]
Market order size	σ_{mo}	[0, 1]
Limit order size	μ_{lo}	[2, 10]
Limit order size	σ_{lo}	[0, 1]
Off-spread relative price	$xmin_{offspr}$	[0, 1]
Off-spread relative price	β_{offspr}	[0, 1]
Crossing limit order	λ_{crs}	[0, 1]
Inside-spread limit order	λ_{inspr}	[0, 1]
Spread limit order	λ_{spr}	[0, 1]
Off-spread limit order	λ_{offspr}	[0, 1]

The output parameters of the sensitivity analysis test are given in Table 8.

Table 8. Output parameters of global sensitivity analysis

Parameter	Symbol
Hurst exponent of volatility	H
Median Autocorrelations of mid-price returns	$R(m)$
Mean first lag autocorrelation term of the order-sign series	$R(o)$
Concave Price Impact	β

As the model is stochastic (agents' actions are defined over probability distributions), there is inherent uncertainty in the range of outputs, even for fixed input parameters. Ten thousand samples from within the parameter space were generated in the following, with the input parameters distributed uniformly in the ranges. Alternatively, for each sample of the parameters' space, the model is run for 100000 trading periods to approximately simulate a trading day on a "high-frequency" timescale.

The global variance sensitivity, as defined in Eq. 23, is presented in Figure 7.

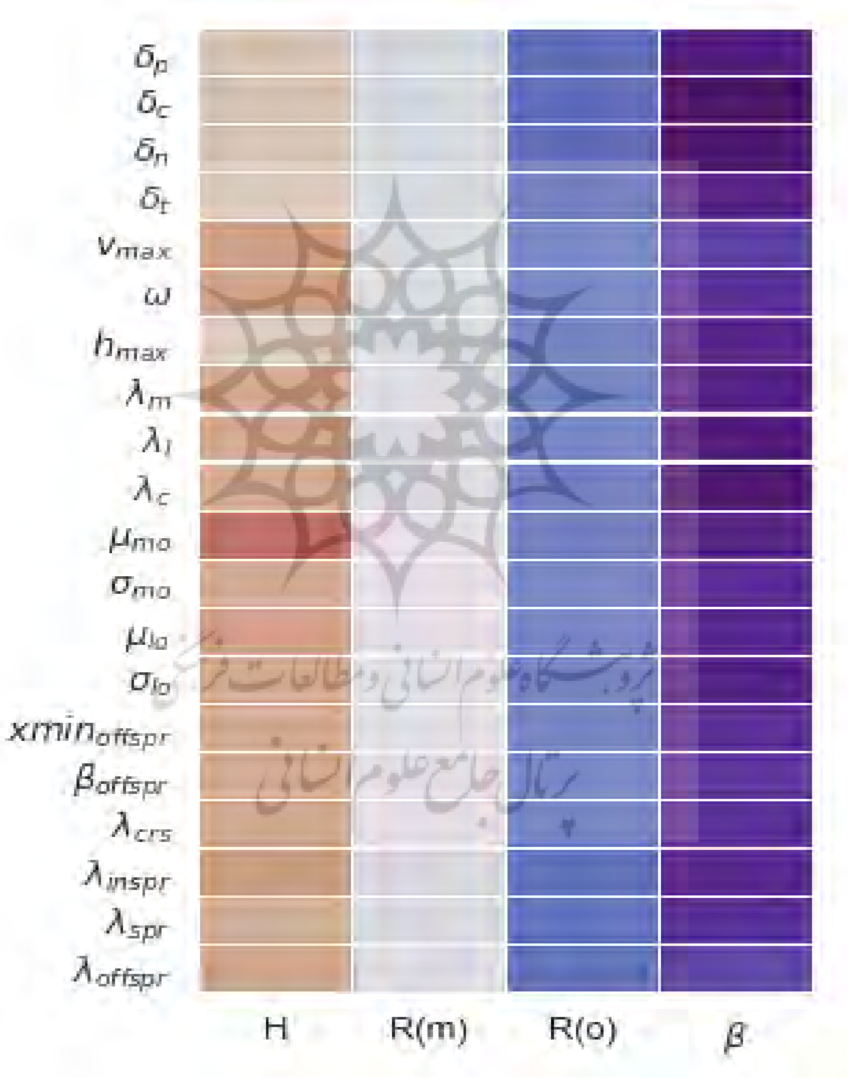

Figure 7. Heatmap of the global variance sensitivity

The global variance sensitivities identify the upper limit of the distribution from which liquidity consumers' order volume is drawn (h_{max}) and the probabilities of each agent group acting (particularly those of the high-frequency traders) as the most important input parameters for all outputs. The most significant influence of these parameters was on the mean first lag autocorrelation term of the order-sign series R(o), followed by the exponent of the price impact function β. β is calculated from Equation (27).

4. Fat-Tailed Distribution of Returns

In this model, only substantial cancellations, orders that fall inside the spread, and large orders that cross the spread can alter the mid-price. This generates many periods with returns of 0, significantly reducing the variance estimate and generating a leptokurtic distribution in the short run.

Figure 8 shows a side-by-side comparison of how the kurtosis of the mid-price return series varies with the lag length for this model.

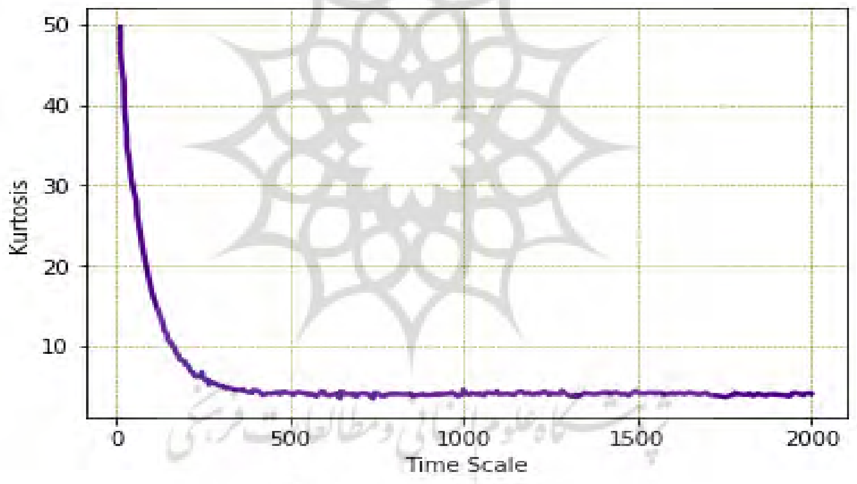

Figure 8. Kurtosis by timescale

Kurtosis is found to be relatively high for short time scales but falls to match levels of the normal distribution at longer time scales, which matches the pattern of decay seen in the empirical data.

5. Volatility Clustering

The Hurst exponent of volatility has been computed using the DFA method described by Peng et al. (1994) to test for volatility clustering. The figure below details the percentage of simulations run with significant volatility clustering defined as $0.6 < H < 1$. Once again, in the shortest time lags, volatility clustering is present at short time scales in all the simulations but rapidly disappears for longer lags. Figure 9 shows the clustering of change volatility by timescale for this model.

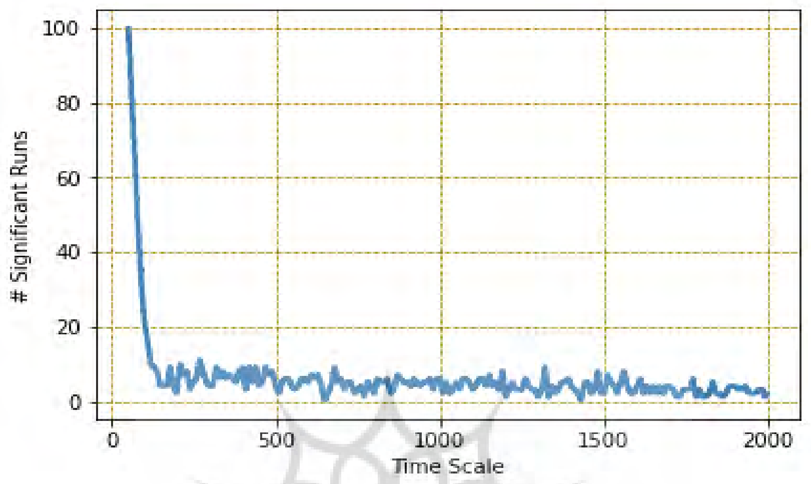

Figure 9. Volatility clustering by timescale

6. Autocorrelation of returns

Table 9 reports descriptive statistics for the first-lag autocorrelation of the return series for our agent-based model:

Table 9. Return autocorrelation statistics

Stats	Min.	Q1	Mean	Q3	Max.
AC mid-price returns	-0.0212	-0.0070	0.0074	0.0216	0.0361
AC trade price returns	-0.3219	-0.1071	0.1059	0.3199	0.5345

There is a weak but significant autocorrelation in the mid-price and trade price returns. This has been empirically observed in other studies and is commonly thought to be due to the refilling effect of the order book after a trade that changes the best price. The result is similar for the trade price autocorrelation, but as a trade price will always occur at the best bid/ask price, a slight oscillation is to be expected and is observed.

7. Long Memory in Order Flow

The descriptive statistics of the order sign series in the model are shown in the table below:

Table 10. Order sign statistics

Stats	Min.	Q1	Mean	Q3	Max.
AC order signs	0.0218	0.0705	0.1191	0.1679	0.2159
H order signs	0.5997	0.6355	0.6713	0.7070	0.7425

In Table 10, H Order signs show a mean Hurst exponent of the order signs time series, which indicates a long-memory process and corresponds with the empirical study results.

8. Concave Price Impact

Figure 10 illustrates the price impact on the model as a function of order size on a log-log scale.

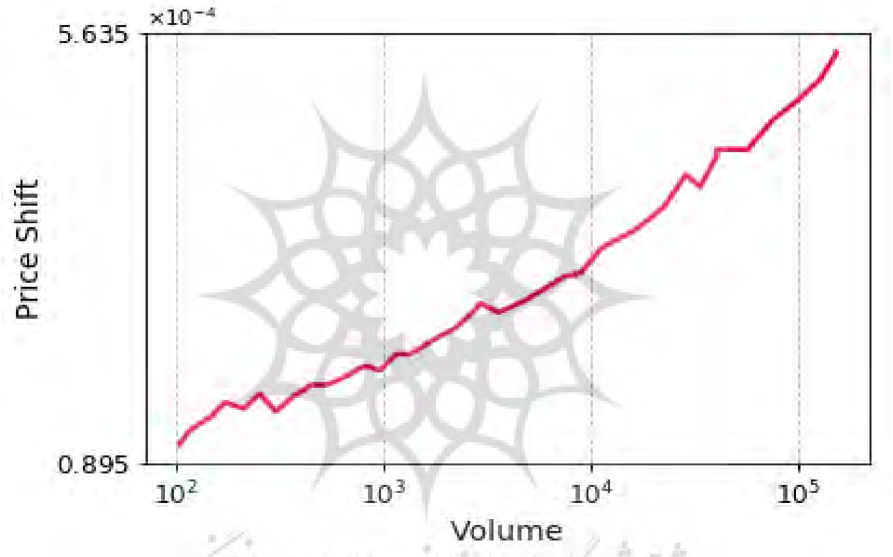

Figure 10. Price Impact

The shape of this curve is very similar to that of the other empirical studies. The price impact is calculated by Equation (25), and for the model is found to be the best fit by the relation $\Delta p \propto v^{31}$, while the empirically measured impact was the best fit by $\Delta p \propto v^{0.35}$. When the market maker's order volume is reduced, the volume at the opposing best price reduces compared to the rest of the book. This allows smaller trades to eat further into the liquidity, stretching the right-most side of the curve.

Figure 11 demonstrates the effects of varying consumers' volume parameter h_{max} and providers' volume parameter v_{max} on the price impact curve.

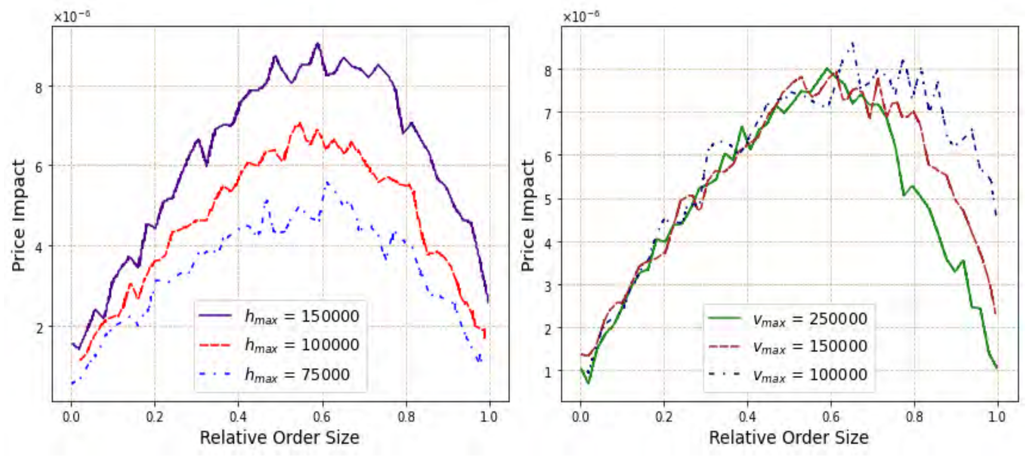

Figure 11. The price impact function with different liquidity consumer and liquidity provider parameterizations

This parameter has little influence on the shape of the price impact function. However, it does affect the size of the impact. Although h_{max} is relatively insensitive to minor changes, when the volume traded by the liquidity consumers is reduced dramatically, the relative amount of available liquidity in the market increases to the point where the price impact is reduced. Similar results are seen as the market makers' order size (v_{max}) increases.

9. Extreme price events

Table 11 shows Flash Crash statistics for the simulated day in the model:

Table 11. Flash crash statistics

Stats	Min.	Median	Mean	Max.
Events per day in ABM	0	1	0.9734	4

In the model, on average, there are 0.9734 events per day, which are very close to the average number observed in empirical data. Such events occur when an agent makes a huge order that eats through the best price (and sometimes further price levels).

Figure 12 shows relative numbers of crash/spike events as a function of their duration.

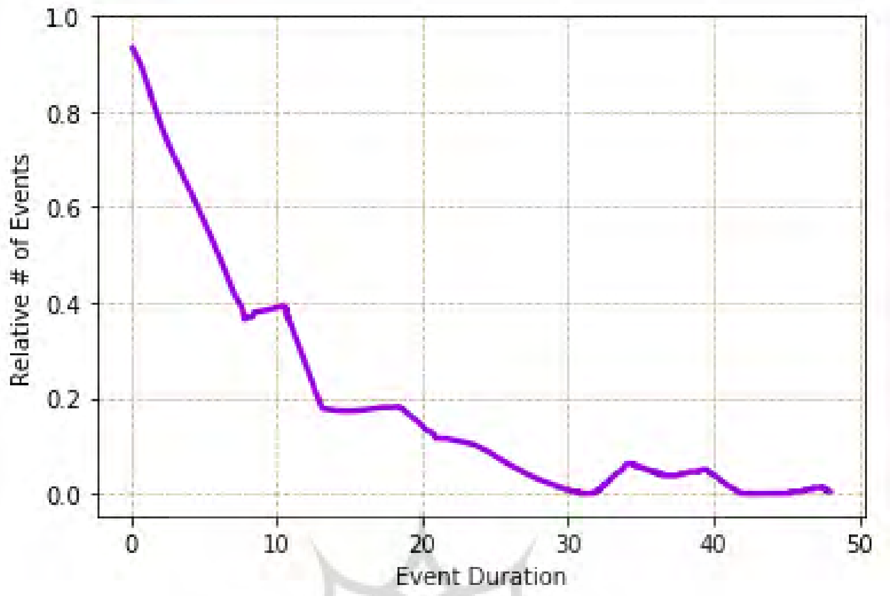

Figure 12. Relative numbers of crash/spike events as a function of their duration

The event duration is the time difference (in simulation time) between the first and last tick in the sequence of jumps in a particular direction. These extreme price events are more likely to occur quickly than over a longer timescale.

Discussion and Conclusion

In this research, machine learning and agent-based modeling methods have been used to discover the driving factors of market price dynamics. The proposed model is designed better to understand the behavior of automated algorithmic trading strategies. The agents are entirely logical and follow simple rules. This is the main feature of any behavioral model, and agent-based models have this feature. Financial markets are an important challenge for agent-based modeling, and one of the most important areas that can show their problem-solving ability. This is because the field has many questions that older approaches cannot solve, and a lot of financial data is available to test the model. The designed model can replicate several well-known statistical features of financial markets, including Volatility Clustering, the autocorrelation of returns, long memory in order flow, concave price impact, and extreme price events. Long memory in the order flow and selection of liquidity behavior of agents plays an important role in the results. This supports the prevailing empirical research results on the microstructure of financial markets. Although the well-known trading strategies in this study are modeled on previous studies, the addition of intelligent agent activity trained using the recurrent neural network adds a new capability to previous models.

It should be noted that extreme price events occur not only due to destructive behaviors that seek to disrupt the market and make a profit after it. However, it can also be due to the interactions of trading strategies. Policymakers must know that focusing efforts to prevent malicious behavior and enforce a regulation in this direction may not be helpful. Instead, market regulators need to focus on understanding how market participants' interactions can lead to unexpected systemic behaviors.

Market impact is an important topic for theoretical and practical research. This cannot only help us understand how information is integrated into market prices, but also help reduce transaction costs. This is especially important for large financial institutions such as pension funds, which have diversified portfolios.

The balance of trading strategies is important in determining price impact performance. In particular, over-activity due to aggressive liquidity-consuming strategies leads to a market where the price effect increases.

The results show that increasing the total number of participants with high frequency does not significantly affect price impact performance. However, the balance of trading strategies is important in determining price impact performance. In particular, over-activity due to aggressive strategies in the liquidity consumer group leads to a market in which the price effect increases.

Policymakers in the financial markets should be aware that focusing efforts to prevent the adverse effects of algorithmic trading and updating regulations in this regard can be helpful. Market regulators must focus on understanding how market participants' interactions can lead to unexpected systemic behaviors.

Studies on algorithmic trading in developed markets, such as those by Jacob Leal et al. (2016), have highlighted the dominance of institutional investors and the efficiency gains from high-frequency trading. However, emerging markets present different challenges and opportunities. Research by Chaboud et al. (2014) and Hendershott et al. (2011) has shown that emerging markets are often characterized by higher volatility, liquidity asymmetry, and a significant proportion of retail investors. These factors can affect the effectiveness of algorithmic trading strategies.

Comparing the performance of the hybrid model with existing research in both developed and emerging markets provides a comprehensive understanding of its applicability. For example, Avellaneda and Stoikov (2008) developed a high-frequency trading model for developed markets, while Gould et al. (2013) studied the statistical properties of limit order books in emerging markets. The paper can highlight its advantages and potential improvements by comparing the hybrid model to these studies.

Similar research shows that integrating agent-based modeling with machine learning leads to more accurate and realistic simulations of complex systems, particularly in financial markets. Kanzari & Ben Said (2023) pointed out that adaptive agents, which can learn and adjust their strategies based on market conditions, are critical for mimicking real-world market dynamics. For example, models populated with adaptive agents were able to reproduce the statistical properties of the S&P 500, especially during periods of crisis .

Including practical implications of the hybrid model in real trading scenarios can enhance the paper's relevance. In addition, addressing limitations and suggesting directions for future research, as recommended by Cont (2001) and Bouchaud (2002), can provide a balanced perspective.

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest concerning the research, authorship and, or publication of this article.

Funding

The authors received no financial support for the research, authorship and, or publication of this article.

References

Alexander, S. S. (1961). Price movements in speculative markets: Trends or random walks. Industrial Management Review (pre-1986), 2(2), 7 .

Aloud, M. E. (2020). The role of attribute selection in Deep ANNs learning framework for high frequency financial trading. Intelligent Systems in Accounting, Finance and Management, 27(2), 43-54 .

Avellaneda, M., & Stoikov, S. (2008). High-frequency trading in a limit order book. Quantitative Finance, 8(3), 217–224 .

Bodie, Z., Kane, A., & Marcus, A. J. (2018). Investments: McGraw-Hill Education.

Booth, A. (2016). Automated algorithmic trading: Machine learning and agent-based modelling in complex adaptive financial markets. University of Southampton ,

Bouchaud, J.-P. (2002). An introduction to statistical finance. Physica A: Statistical Mechanics and its Applications, 313(1-2), 238-251 .

Calvez, B., & Hutzler, G. (2005). Automatic tuning of agent-based models using genetic algorithms. Paper presented at the International workshop on multi-agent systems and agent-based simulation.

Chaboud, A. P., Chiquoine, B., Hjalmarsson, E., & Vega, C. (2014). Rise of the machines: Algorithmic trading in the foreign exchange market. The Journal of Finance, 69(5), 2045-2084 .

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

Crooks, A., Heppenstall, A., Manley, E., & Malleson, N. (2018). Agent-based modelling and geographical information systems: a practical primer .

Dixon, M. (2017). Sequence classification of the limit order book using recurrent neural networks. Journal of Computational Science. doi:https://doi.org/10.1016/j.jocs.2017.08.018

Dixon, M. F. (2020). Machine Learning in Finance: from Theory to Practice: Springer Nature.

Donadio, S., & Ghosh, S. (2019). Learn Algorithmic Trading: Build and Deploy Algorithmic Trading Systems and Strategies Using Python and Advanced Data Analysis: Packt Publishing.

Evensen, G. (2003). The ensemble Kalman filter: Theoretical formulation and practical implementation. Ocean dynamics, 53, 343–367 .

Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. The Journal of Finance, 25(2), 383-417 .

Fama, E. F., & Blume, M. E. (1966). Filter rules and stock-market trading. The Journal of Business, 39(1), 226–241 .

Frazier, P. I. (2018). A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811 .

Gould, M. D., Porter, M. A., Williams, S., McDonald, M., Fenn, D. J., & Howison, S. D. (2013). Limit order books. Quantitative Finance, 13(11), 1709-1742 .

Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850 .

Guo, T. X. (2005). An agent-based simulation of double-auction markets: University of Toronto.

Heaton, J. B., Polson, N. G., & Witte, J. H. (2017). Deep learning for finance: deep portfolios. Applied Stochastic Models in Business and Industry, 33(1), 3–12 .

Hendershott, T., Jones, C. M., & Menkveld, A. J. (2011). Does algorithmic trading improve liquidity? The Journal of Finance, 66(1), 1-33 .

Hirsa, A. (2013). An introduction to the mathematics of financial derivatives: Academic press.

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: A field guide to dynamical recurrent neural networks. IEEE Press.

Huang, Z.-F., & Solomon, S. (2001). Power, Lévy, exponential and Gaussian-like regimes in autocatalytic financial systems. The European Physical Journal B-Condensed Matter and Complex Systems, 20(4), 601–607 .

Jacob Leal, S., Napoletano, M., Roventini, A., & Fagiolo, G. (2016). Rock around the clock: An agent-based model of low-and high-frequency trading. Journal of Evolutionary Economics, 26, 49-76 .

Johnson, B. (2010). Algorithmic Trading & DMA: An introduction to direct access trading strategies (Vol. 1): 4Myeloma Press London.

Johnson, N., Zhao, G., Hunsader, E., Qi, H., Johnson, N., Meng, J., & Tivnan, B. (2013). Abrupt rise of new machine ecology beyond human response time. Scientific reports, 3(1), 1–7 .

Kanzari, D., & Said, Y. R. B. (2023). A complex adaptive agent modeling to predict the stock market prices. Expert Systems with Applications, 222, 119783 .

Kim, G.-r., & Markowitz, H. M. (1989). Investment rules, margin, and market volatility. Journal of portfolio management, 16(1), 45 .

Levy, M., Levy, H., & Solomon, S. (1994). A microscopic stock market model: cycles, booms, and crashes. Economics Letters, 45(1), 103–111 .

Lillo, F., & Farmer, J. D. (2004). The long memory of the efficient market. Studies in nonlinear dynamics & econometrics, 8 .(3)

Lillo, F., Farmer, J. D., & Mantegna, R. N. (2003). Master curve for price-impact function. Nature, 421(6919), 129–130 .

McGroarty, F., Booth, A., Gerding, E., & Chinthalapati, V. R. (2019). High frequency trading strategies, market fragility and price spikes: an agent based model perspective. Annals of Operations Research, 282(1), 217–244 .

Oesch, C. (2014). An agent-based model for market impact .

Peng, C. K., Buldyrev, S. V., Havlin, S., Simons, M., Stanley, H. E., & Goldberger, A. L. (1994). Mosaic organization of DNA nucleotides. Physical review e, 49(2), 1685 .

Samanidou, E., Zschischang, E., Stauffer, D., & Lux, T. (2007). Agent-based models of financial markets. Reports on Progress in Physics, 70(3), 409 .

Sobol, I. M. (2001). Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. Mathematics and computers in simulation, 55(1-3), 271-280 .

Taylor, G. W., & Hinton, G. E. (2009). Factored conditional restricted Boltzmann machines for modeling motion style—paper presented at the Proceedings of the 26th annual international conference on machine learning.

van der Hoog, S. (2017). Deep Learning in (and of) Agent-Based Models: A Prospectus. arXiv preprint arXiv:1706.06302 .

Wilensky, U., & Rand, W. (2015). An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo: MIT Press.

Bibliographic information of this paper for citing:

Poostforoush, Mohammad Hossein; Monajemi, Amirhassan; Daei-Karimzadeh, Saeed & Samadi, Saeed (2025). Automation of Algorithmic Trading Strategies in Artificial Financial Markets by Combining Machine Learning Techniques and Agent-based Modeling. *Iranian Journal of Finance*, 9(3), 95-134.