Received March 4, 2021, accepted March 16, 2021, date of publication March 23, 2021, date of current version April 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3068269

Improving Generalization in Reinforcement Learning–Based Trading by Using a Generative Adversarial Market Model

CHIA-HSUAN KUO[1], CHIAO-TING CHEN[2], SIN-JING LIN[1], AND SZU-HAO HUANG[3], (Member, IEEE)

[1]Institute of Information Management, National Chiao Tung University, Hsinchu 30010, Taiwan
[2]Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan
[3]Department of Information Management and Finance, National Chiao Tung University, Hsinchu 30010, Taiwan

Corresponding author: Szu-Hao Huang (szuhaohuang@nctu.edu.tw)

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Contract MOST 110-2622-8-009 -014 -TM1, Contract MOST 109-2221-E-009 -139, Contract MOST 109-2622-E-009 -002 -CC2, and Contract MOST 109-2218-E-009 -015; and in part by the Financial Technology (FinTech) Innovation Research Center, National Yang Ming Chiao Tung University.

ABSTRACT With the increasing sophistication of artificial intelligence, reinforcement learning (RL) has been widely applied to portfolio management. However, shortcomings remain. Specifically, because the training environment of an RL-based portfolio optimization framework is usually constructed based on historical price data in the literature, the agent potentially 1) violates the definition of a Markov decision process (MDP), 2) ignores their own market impact, or 3) fails to account for causal relationships within interaction processes; these ultimately lead the agent to make poor generalizations. To surmount these problems—specifically, to help the RL-based portfolio agent make better generalizations—we introduce an interactive training environment that leverages a generative model, called the limit order book-generative adversarial model (LOB-GAN), to simulate a financial market. Specifically, the LOB-GAN models market ordering behavior, and LOB-GAN's generator is utilized as a market behavior simulator. A simulated financial market, called Virtual Market, is constructed by the market behavior simulator in conjunction with a realistic security matching system. Virtual Market is then leveraged as an interactive training environment for the RL-based portfolio agent. The experimental results demonstrate that our framework improves out-of-sample portfolio performance by 4%, which is superior to other generalization strategies.

INDEX TERMS Artificial market simulation, portfolio management, reinforcement learning.

I. INTRODUCTION

Portfolio management, a long-established part of quantitative trading, has the aim of satisfying a predefined utility function by continually reallocating capital across some set of financial products. Methods of portfolio management come in one of three types: 1) traditional methods (such as the momentum [1] and contrarian strategies [2]), 2) machine-learning methods (such as pattern matching [3]), and 3) reinforcement learning (RL)-based methods [4], [5]. With the thriving development of deep neural networks, many researchers have integrated deep learning with RL to achieve impressive performance in several domains of finance, such as forex trading [6], portfolio management [4], [5], [7], [8], and market making [9].

Most successful RL studies have used realistic physics engines or dynamic interactive entities to construct the training environment. For example, AlphaZero [10] trains an agent to play a board game through self-play. Here, self-play means that the environment that the agent faces, which means the player agent against, is generated by the best player (agent), which is trained by the neural networks, from all previous iterations. The training agent obtains continual feedback in response to its own actions, resulting in a robust and reasonable interrelationship between the training environment and agent. Research on RL-based portfolio management, however, has been less successful. In such research, historical price data remain straightforwardly used to construct the training environment [4], [5], [7], [8]. From the agent's

The associate editor coordinating the review of this manuscript and approving it for publication was Constantinos Marios Angelopoulos.

perspective, feedback from such a training environment is not responsive to the agent's actions. Consequently, the agent faces several problems when optimizing its actions in relation to this unresponsive training environment. First, the state obtained from the environment is unrelated to the agent's action. The agent's interactions with this unresponsive environment potentially violate the definition of a Markov decision process (MDP)—where the MDP theorem explicitly defines a state transition as one that depends on the current state and action. Because the MDP theorem is the fundamental theorem of RL, violating the definition of an MDP renders the optimization process of the RL-based portfolio agent unreasonable. Second, such unresponsiveness means that the environment cannot render an appropriate market reaction in response to the agent's action. In other words, an environment constructed based on historical price data cannot simulate the agent's impact on the market. Thus, an agent optimized using historical price data may produce poor generalizations: the trading knowledge constructed from data in the in-sample period (in training) fails to apply in the out-of-sample period (in testing). Regardless of how well a model fits the training data, a model that generalizes poorly is useless for solving practical decision-making problems. Thus, generalizability can be considered the largest obstacle that must be surmounted in the construction of RL-based portfolio management models. Studies have employed data augmentation and adversarial attack or adversarial training [9], [11] to improve the generalization ability of RL-based trading agents by injecting randomization into the environment. However, these studies have mostly used historical price data to construct the environment; the injection of random noise does not directly address the aforementioned problems.

In our opinion, two types of solutions can be used to address the aforementioned problems. The first is interacting the RL-based portfolio agent with real stock exchange data for portfolio optimization. The second is using another AI model to construct a realistic virtual market for the RL agent to interact with. The first solution bases rewards on transaction results in a real financial market. However, because this solution is costly and requires a relatively long data-collection time for the agent to converge, it cannot be realistically applied to RL-based portfolio optimization. The second approach is where our main contribution lies. In our study, a variation of the generative adversarial network (GAN) is proposed to simulate market ordering behavior by modeling the distribution of the historical limit order. The generative model is then used to construct a synthetic stock exchange as a training environment for the agent. The proposed learning framework enables the agent to obtain the simulated market's reaction to their trading decision. By doing so, the causal interrelationship between the state and action is enhanced. Furthermore, because the agent is allowed to be involved with the state transition process, the simulated stock exchange can prevent the agent from violating the definition of an MDP; this makes the use of RL in portfolio optimization reasonable by ensuring that the fundamental theorem underpinning the RL framework holds. By interacting with the simulated stock exchange, the agent is able to explore a greater range of previously unforeseen market situations; the training data set is also substantially more diverse. To the best of our knowledge, this is the first study to use a generative model to reconstruct a financial market in a simulation for RL-based portfolio management with the purpose of improving the generalization ability of the agent. The major contributions of this study are listed as follows:

- A proposed generative model, called limit order book (LOB)-GAN, models the distribution that underlies the historical limit order. LOB-GAN is used to simulate the ordering behavior of the overall investors in the market.
- A limit-order transform module is introduced to allow the LOB-GAN to synthesize the relative order quantity instead of directly predicting the order price and corresponding quantity.
- A synthetic stock exchange, called Virtual Market, is constructed by having the generator within the LOB-GAN cooperate with a security matching system. Virtual Market can render a simulated market reaction based on the agent's trading decision.
- A novel RL-based portfolio optimization learning framework that leverages Virtual Market is proposed. This framework ensures that the definition of an MDP is never violated by enabling a closer interrelationship between the action and transition state.

The remaining parts of the paper are organized as follows: Section II reviews the literature; Section III states the hypotheses and defines the problem; Section IV introduces the proposed market behavior simulator, the construction of Virtual Market, and other generalization strategies; Section V introduces the proposed RL-based portfolio optimization framework; Section VI presents the experimental results; and Section VII concludes the paper and discusses future research directions.

II. LITERATURE REVIEW

This section reviews three bodies of literature: that on leveraging RL in finance, on RL generalization techniques, and on artificial market simulation.

A. REINFORCEMENT LEARNING IN FINANCE

RL has been widely applied in several domains of finance, such as market making and forex trading, and it is especially important in portfolio management. In this section, we focus on reviewing the literature on RL-based portfolio management. Empirically, portfolio management can be separated into three major steps: portfolio selection, weighting, and rebalancing. In portfolio selection, the focus is on selecting portfolio assets; in portfolio weighting, the process decides capital assignment; and, in portfolio rebalancing, a decision is made on whether and when to change the portfolio weight. Sbruzzi *et al.* [12] focus on portfolio selection and use an RL framework in which an agent for asset pool selection optimizes the selection strategy. Wang *et al.* [4] bridged the processes of portfolio selection and weighting by using their proposed AlphaStock method. Specifically, the authors formulated the Specialized Cross-Asset Attention Network (CAAN) mechanism in AlphaStock to capture the interrelationship within portfolio assets. Jiang *et al.* [7] focused on portfolio weighting and proposed their Ensemble of Identical Independent Evaluators (EIIE) topology. Their portfolio selection strategy straightforwardly uses the trading volume as the basis, and transaction cost (a critical issue in the execution of algorithmic trading strategies) is accounted for in their learning framework. The authors examined several time-series feature extraction models using their EIIE topology. Shi *et al.* [5] extended the EIIE topology in their Ensemble of Identical Independent Inception (EIII) topology, which leverages an inception network to simultaneously consider different scales of price movement. Their experimental results demonstrated that the EIII topology yields better portfolio performance than the original EIIE does. Ye *et al.* [8] also extended the EIIE topology in their State-Augmented RL (SARL) topology, in which cooperation is introduced into a heterogeneous data set to help the agent make better predictions. Tang *et al.* [13] also emphasize combining multiple sources, where traditional indicators and the module of a pretrained GAN each constitute a distinct stream of data. Li *et al.* [14] applied a novel RL algorithm that utilizes stacked denoising autoencoders (SDAEs) to construct an agent with the aim of obtaining a robust state representation. Despite these advances, research on RL-based portfolio optimization has mostly used historical data to optimize the agent, which potentially results in an agent with poor generalization ability.

B. GENERALIZATION IN REINFORCEMENT LEARNING

The generalization problem in RL has been studied in various domains. Whiteson *et al.* [15] divided the generalization problem into within-task and out-of-task variants. In the within-task variant, generalization ability is satisfactory if the agent that was optimized on training trajectories performs well on testing trajectories from the same environment. In the out-of-task variant, generalization ability is satisfactory when the agent performs well in an environment that differs from the training environment. The methods that have been used to address the generalization problem in RL can be divided into five categories.

- *Regularization Approach:* Several techniques, such as dropout and L2 regularization, are applied to prevent the agent from overfitting in the limited state space [16]. Igl *et al.* [17] propose selective noise injection (SNI), which preserves the regularizing effect but mitigates the side effect on the gradient for greater adaptability to RL.
- *Adversarial Training:* Different settings of the perturbation generation strategy are introduced in RL-based trading [9], [11]. The injected noise can 1) help the agent to learn how to furnish a robust representation and 2) diversify the training environment.
- *Data Augmentation:* To make the data more diverse, a transformation is applied on the state [18], [19].
- *Transfer Learning:* By focusing on helping the agent generalize to a new task, it is widely used for domain adaption [20]. Gamrian and Goldberg [21] further utilize GAN to map visual observations from the target to source domains.
- *Meta-Learning:* The agent learns a metapolicy that helps it quickly adapt to other domains [22]. Wang *et al.* [23] also focus on the problem of making the agent rapidly adapt to new tasks; they do so by extending a recurrent network to support meta-learning in RL.

In this study, we focus on the within-task generalization ability of the agent, where the goal of the agent is to learn a general trading strategy that yields comparable portfolio performance between the testing and training periods. This goal is similar to that in [9], [11]. However, similar to research on RL in finance, research on improving generalization in finance has been based on historical price–based training environments. Therefore, the aforementioned problems of using historical data remain unsolved in the literature.

C. ARTIFICIAL MARKET SIMULATION

Researchers have long attempted to model investor behavior. Pioneering studies have focused on the potential of the efficient market hypothesis (EMH) [24], which holds that people are always rational enough to make the best decision. However, other researchers have found that people do make irrational decisions, such as under the herd effect [25]. Behavioral economics was thus proposed to model such irrationality. Recent studies have focused on behavioral prediction. According to Lovric *et al.* [26], investment decisions can be modeled as an outcome of investor–environment interactions. Studies have also proposed several interdependent variables, such as time preference, risk attitude, and personality, that influence the investment process. Furthermore, in the framework proposed by Shantha *et al.* [27], investors learn from their trading experiences (individual learning) or by imitating others (social learning).

Artificial market simulation enables researchers to construct situations that cannot be captured in historical data. Such simulations are thus widely used to analyze various issues in finance, such as short selling regulations [28], transaction taxes [29], and the speed of order matching systems [30]. Agent-based simulation, which incorporates multiple agents to reproduce stylized facts in a real market, is the most common technique in artificial market simulation. The simulation process comprises several parts. First, the intelligence levels, utility function, and learning ability of the involved agents are defined [31]. Second, the asset price is determined [32]. Third, the type and number of traded assets involved in artificial market construction are declared [33]. Fourth, the learning process, which is highly related to the agent's level of intelligence, is determined [34], [35]. Fifth, and finally, the simulated market is calibrated and validated. Specifically, calibration is the selection of parameters that make the simulated market behave closest to a real market, and validation pertains to whether the simulated market behaves as a real market does. In addition to using an agent-based model to construct a simulated market, Li *et al.* [36] proposed Stock-GAN to produce limit-order data at high fidelity to support market design and analysis in continuous trading systems. In this study, we utilized a generative model to construct a financial market. We not only reconstructed a financial market with a realistic pricing mechanism but also combined the simulated market with the RL trading agent. By synthesizing the simulation of markets with the RL-based portfolio optimization framework, we overcame the aforementioned drawbacks of using historical price data for agent optimization.

III. PRELIMINARIES

This section states the hypotheses, discusses this study's limitations, and formulates the problem of applying RL in portfolio management.

A. HYPOTHESES

We propose a generative model to simulate the market's reaction to the agent's action. The following assumptions must therefore be made:

- Because the simulated financial market is responsible for generating a reasonable reaction to the agent's action, the agent is assumed to have the ability to influence the behavior of other investors in the market.
- The ordering behavior of investors sufficiently reflects the influence of exogenous variables on the financial market. Therefore, we only modeled market ordering behavior when synthesizing the reasonable market reaction.

This study has another limitation in addition to these assumptions. Because we still lack a systematic approach for verifying the authenticity of the generated limit order, the evaluation of portfolio performance in a simulated financial market may expose the agent to the risk of unrealistic estimations. Thus, we used historical price data to evaluate generalization ability.

B. PROBLEM DEFINITION

Portfolio management is a decision-making process in which funds are continually reallocated to different assets. The process of portfolio strategy making can be formulated as an MDP. The MDP is represented as a tuple $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, p_0, \gamma>$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}$ the state transition function, $\mathcal{R}$ the reward function, p_0 the probability distribution of the initial state, and $\gamma \in [0, 1)$ the reward discount factor. In the case of portfolio management, the agent aims to find an optimal policy $\pi(a \mid s)$, where the action $a \in \mathcal{A}$ is optimal with respect to state $s \in \mathcal{S}$. In this optimal policy, the expected

TABLE 1. Annotation table.

Symbol	Meaning	Symbol	Meaning
$\mathcal{S}$	state set	z	noise
$\mathcal{A}$	action set	ω	discriminator parameters
$\mathcal{P}$	transition function	θ	generator parameters
$\mathcal{R}$	reward	X_t	market state
γ	reward discount factor	U_t	self state
π	price at time i	V_t^f	price movement vector
$\mathcal{O}$	order stream	w_t	portfolio weight
T	time	C	cash initial hold
α	learning rate of LOB-GAN	ψ	switch remainder factor
m	window size	ϕ	policy parameters
n	generate step	J	objective function
d	batch size	τ	trajectory
D	discriminator	η	learning rate of policy network
G	generator	$\mathcal{B}$	current active order
k	number of G steps per D	$\mathcal{Q}$	price database
x	sampled limit order	q	asset number
y	condition	M	epoch

return is maximized:

$$\pi^* = \arg\max \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\right], \tag{1}$$

where $s_0 \sim p^0,\ a_t \sim \pi(\cdot \mid s_t), and\ s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)$. The RL-based portfolio management framework mainly contains an environment and an agent. The mapping from MDP to the learning framework is described as follows.

1) ENVIRONMENT

The design of the environment comprises the following elements: the (1) state $s_t \in \mathcal{S}$, which contains the trading status of the agent or a period of the environment-provided price sequence; (2) state Transition $\mathcal{P}(\cdot \mid s_t, a_t)$, which renders the next state s_{t+1} given a previous state and action; and (3) reward function $\mathcal{R}(s_t, a_t)$, which is the utility function that defines the portfolio performance of the agent and serves as the objective function for the agent to maximize.

2) AGENT

The agent is a portfolio manager that decides the portfolio weight for each asset to maximize the final reward. The design of the agent includes the following two elements. The first is the action $a_t \in \mathcal{A}$; the action corresponds to the portfolio weight, which indicates the assignment of capital. To fulfill the weighting requirement, the action space is set to be continuous. The second is the policy $a_t \sim \pi(\cdot \mid s_t)$; the optimal policy is one in which the expected reward is maximized for a given observation s_t.

The important annotations we use in our framework could be found in Table 1.

IV. VIRTUAL MARKET DESIGN

In this section, a limit-order transform module is introduced. This module transforms the limit-order stream into a relative representation, and the market behavior simulator subsequently uses the relative limit-order stream as a condition to generate the limit-order quantity. The market behavior simulator is realized through a naive supervised learning model and the proposed LOB-GAN. Virtual Market was

Algorithm 1 Limit-Order Transform Module

```
Require: Vector shifting tool $Shift(vector, unit)$
Require: Anchor time step $t$
Input: Bid limit-order stream $O^{bid} = \{O_i^{bid}\}_{i=0}^{T}$, ask limit-order stream $O^{ask} = \{O_i^{ask}\}_{i=0}^{T}$, historical strike price $O^{stk} = \{O_i^{stk}\}_{i=0}^{T}$
1: function $Trans$ $(O^{bid}, O^{ask}, O^{stk})$
2:     for $i = 0, \dots, T$ do
3:         $unit \leftarrow$ Get price level between $O_i^{stk}$ and $O_t^{stk}$
4:         $O_i^{bid} \leftarrow Shift(O_i^{bid}, unit)$
5:         $O_i^{ask} \leftarrow Shift(O_i^{ask}, unit)$
6:     return $O^{bid}, O^{ask}$
```

Algorithm 2 Market Behavior Model (Supervised Learning)

```
Require: Learning rate $\alpha$, window size $m$, predict window size $n$, batch size $d$.
Require: Initial model $\mu$ parameters $\varphi$.
Require: Limit-order transform module $Trans$
1: for number of training steps do
2:     Sample a mini-batch of limit-order sequence $x$
3:     $\{O^{bid}, O^{ask}, O^{stk}, O^{time}\} \leftarrow x$
4:     $\{\tilde{O}^{bid}, \tilde{O}^{ask}\} \leftarrow Trans(O^{bid}, O^{ask}, O^{stk})$
5:     $x \leftarrow \{\tilde{O}^{bid}, \tilde{O}^{ask}, O^{stk}, O^{time}\}$
6:     Get target $y \leftarrow x_{t=t}^{t-m+1}$ and condition $\hat{x} \leftarrow x_{t=t}^{t+n}$
7:     Update $\mu$ parameter $\varphi$ to minimize:
       $MSE = \frac{1}{d}\sum_{i=1}^{d}(\mu(\hat{x}) - y)^2$
       $\varphi \leftarrow \varphi - \alpha \nabla MSE(\mu)$
```

constructed by combining the pretraining market behavior simulator with a realistic security matching system. Each component is detailed in turn in the following subsections.

A. LIMIT-ORDER TRANSFORM MODULE

Because the goal of a market behavior simulator is to generate the relative limit-order quantity, the limit-order transform module is required to transform the limit-order stream into a relative representation. The relative representation is obtained by calculating the price level between a time step and the user-defined anchor time step. The limit-order transform module operates independently for each stock; this is because the minimum tick increment of price quotes differs with the price scale and between each financial product. Algorithm 1 formally defines the steps constituting the limit-order transform module. Specifically, the vector shifting tool *Shift* is used to shift the element in *vector* by a given offset index *unit*. The sequential T time steps of a limit-order quantity exceeding ± 10 of the price level are denoted as $\{O_i^{bid}\}_{i=0}^{T}$ and $\{O_i^{ask}\}_{i=0}^{T}$ on the bid and ask sides, respectively. The sequential T time step of a historical strike price is denoted as $\{O_i^{stk}\}_{i=0}^{T}$. The historical strike price at a selected anchor time step t is leveraged as the base price of a current round of a transform operation. At each time step over period T, the offset index *unit* is calculated per the minimum tick increment rule. The whole process should be executed each time before the feeding limit-order streams into the market behavior simulator as a conditional input. By executing the limit-order transform procedure, the sequence of a bid/ask limit-order stream can be transformed into another form, and it is interpreted as the relative representation based on the anchor strike price O_t^{stk}. The limit-order transform module is a crucial component of the market behavior simulator that allows the simulator to model the relative limit-order quantity distribution. In the subsequent subsection, the cooperating transform module and market behavior simulator are detailed.

B. MARKET BEHAVIOR SIMULATOR

Because the stock market is not always efficient, it is possible to model its market behavior to earn a profit. The market behavior simulator is used to synthesize the limit order and can be realized through two distinct approaches: naive supervised learning and GAN. The inputs are the same for each approach: the bid order log, ask order log, and strike price. A model without historical strike prices can successfully generate the bid-ask order, but investors typically check the stock price if they wish to long or short the stock. To imitate the condition, we add the historical strike price into our model. The models of the market behavior simulator for the two approaches are introduced as follows.

1) NAIVE SUPERVISED LEARNING

Under the naive supervised learning setting, the goal of the market behavior simulator is to generate limit orders that are most similar to the historical limit order in the data. The model must be pretrained per Algorithm 2, which proceeds as follows. First, sample a minibatch x of a limit order stream from the historical limit-order data; then, convert each stream individually into its relative representation using the limit-order transform module $Trans(\cdot)$. The condition $\hat{x}$ and prediction target y can be retrieved from x. The model μ uses the previous m time step from t within the limit-order stream as the input feature and predicts the relative limit-order quantity distribution of the next n time step. The generative model μ is set to minimize the difference between the generated limit-order quantity $\mu(\hat{x})$ and the target limit-order quantity y in terms of the mean squared error. By using naive supervised learning to construct the market behavior simulator, the generative model emphasizes the direct minimization of the difference from historical data. Therefore, the closer the fit to the historical data, the better the generative model is. The diversity of the generated limit order may be reduced when the generative model aims at merely fitting to historical data. Therefore, we introduce another market behavior simulator that is derived from GAN.

2) LOB-GAN

We propose the LOB-GAN as an alternative approach to generating a more diverse limit order. The idea underlying LOB-GAN is similar to that underlying the Stock-GAN

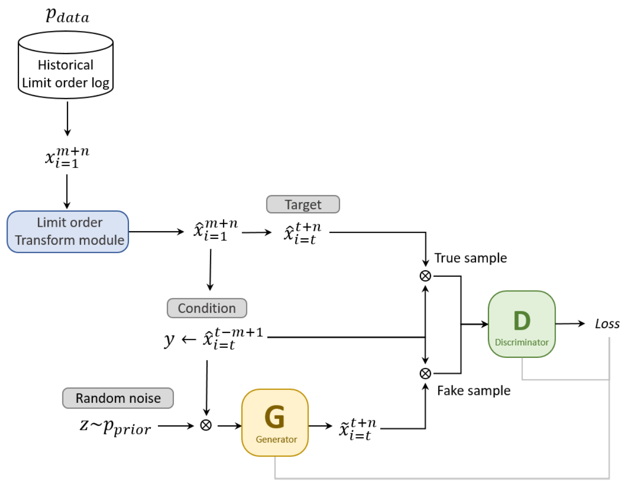


FIGURE 1. **LOB-GAN architecture.**

proposed by Li *et al.* [36]. Stock-GAN aims to generate the order arrival time, order price, order quantity, and order type of the limit order at the next time step. However, in Stock-GAN, the generator is forced to generate a large quantity of heterogeneous information, and the rationality of doing so is debatable, especially for price prediction tasks. Therefore, the LOB-GAN improves the limit-order generation task by introducing the limit-order transform module, which eliminates the challenge faced in price prediction tasks. Our proposed LOB-GAN focuses on generating a relative limit-order quantity distribution instead of directly generating the limit-order price and corresponding quantity. The LOB-GAN architecture is shown in Fig. 1.

The LOB-GAN framework follows a vanilla GAN setting, which comprises a generator G and discriminator D. To model the historical dependence within a sequential limit-order stream, a Conditional Generative Adversarial Net (CGAN) [37] is then introduced into the structure of the LOB-GAN. The Wasserstein divergence and several optimization techniques used in WGAN [38] are also applied to the LOB-GAN. The generator of LOB-GAN is responsible for producing a certain period in which the relative price level is $+10$ and -10 for the bid and ask limit order, respectively. The generator utilizes the previous m steps of the limit-order stream as an auxiliary condition to generate the relative limit-order quantity for the following n time steps. The target of the generator is to fool the discriminator by minimizing

$$-\mathbb{E}_{y\sim p_{data},z\sim p_{prior}} D(G(z \mid Trans(y)) \mid Trans(y)), \qquad (2)$$

where p_{data} is the distribution of the market behavior underlying the historical limit order; p_{prior} is the Gaussian distribution that is used to sample random noise z as one of many inputs for the generator; y comprises time interval information and limit-order stream data $\{\tilde{O}_i^{bid}, \tilde{O}_i^{ask}, O_i^{stk}\}_{i=1}^m$, indexed sequentially by order time; and $Trans(\cdot)$ is the limit-order transform module. The terms $\tilde{O}^{bid}$ and $\tilde{O}^{ask}$ represent the limit-order stream after the transformation process. The time interval information is incorporated as a condition because the ordering behaviors of investors vary between time intervals. By minimizing Equation 2, reinforcement is used to help the generator learn how to execute a generative process that can produce a realistic limit-order stream. With regard to the network structure, for the generator, the inputs are timesteps, asset prices, and order logs. We use two convolutional layers for learning the embedding of the bid–ask conditions. Subsequently, the condition embedding and all inputs are concatenated, and a five-layer fully connected neural network is passed. Finally, the generator outputs the bid–ask state at the next timestamp.

By contrast, the mission of the discriminator is to distinguish the generated limit order from the real one by the given previous m steps of the limit-order stream as the condition. The discriminator can be interpreted as a type of scoring function that aims to give a higher score to a real sample and a lower score to a fake sample. The objective of the discriminator is to maximize

$$\mathbb{E}_{x,y\sim p_{data}} D(x \mid Trans(y)) \\ - \mathbb{E}_{y\sim p_{data},z\sim p_{prior}} D(G(z \mid Trans(y)) \mid Trans(y)), \qquad (3)$$

where the term to the left of the minus sign aims to give a higher score to the real sample, and the term to the right of the minus sign aims to minimize the score given to the generated sample. Crucially, the sampled limit order x must be time-dependent in relation to condition y. The discriminator also takes timesteps, asset prices, and the order log as its inputs, which are processed by three convolutional layers and four fully connected layers that subsequently output the sample's probability of validity.

The entire LOB-GAN learning process is summarized in Algorithm 3. The generator and discriminator are enhanced iteratively against each other. The adversarial learning process listed in Algorithm 3 is the pretraining phase of the LOB-GAN. Under the LOB-GAN setting, the target of the generative model is different from that under naive supervised learning. Instead of fitting to historical limit-order data, the generative model within LOB-GAN aims to model the discriminator-made distribution underlying the historical limit-order data from the critic.

From line[8] to line[12], the generator is fixed and the discriminator is first optimized through a minibatch d of the sequential historical limit-order stream. Each limit-order sequence is sent into the transform module *Trans* to individually obtain relative representations. The condition y is retrieved from the transformed limit-order stream and fed into the generator to generate a fake sample. By using the true sample, generated sample, and conditional input, the discriminator learns to justify a given sample through Equation 3. Subsequently, the discriminator is fixed, and the generator is optimized per line[14] to line[18]. A minibatch of limit-order streams is resampled, and input preprocessing under similar conditions is conducted. The generator is optimized with Equation 2 to defeat the current discriminator.

Algorithm 3 LOB-GAN Pre-Training Process

```
Require: Learning rate $\alpha$, window size $m$, generate step $n$, batch size $d$, number of generator steps per discriminator iteration $k$, limit-order transform module $Trans$.
Require: Initial discriminator $D$ parameters $\omega$. Initial generator $G$ parameters $\theta$.
1:  function TransLimitOrder($y$)
2:      for $j = 1, \ldots, d$ do
3:          $\{O^{bid}, O^{ask}, O^{stk}, O^{time}\} \leftarrow y^{(j)}$
4:          $\{\tilde{O}^{bid}, \tilde{O}^{ask}\} \leftarrow Trans(O^{bid}, O^{ask}, O^{stk})$
5:          $y^{(j)} \leftarrow \{\tilde{O}^{bid}, \tilde{O}^{ask}, O^{stk}, O^{time}\}$
6:      return $y$
7:  for number of training steps do
8:      Sample $\{x^{(j)}\}_{j=1}^{d} \sim p_{data}$ a batch of real data.
9:      Sample $\{z^{(j)}\}_{j=1}^{d} \sim p_{prior}$ a batch of noise.
10:     Get condition $\{y^{(j)}\}_{j=1}^{d}$ from $\{x^{(j)}\}_{j=1}^{d}$
11:     $\{y^{(j)}\}_{j=1}^{d} \leftarrow$ TransLimitOrder($\{y^{(j)}\}_{j=1}^{d}$)
12:     Update $D$ parameter $\omega$ to maximize:
        $V = \sum_i D(x^{(i)} \mid y^{(i)}) - \sum_i D(G(z^{(i)} \mid y^{(i)}) \mid y^{(i)})$
        $\omega \leftarrow \omega + \alpha \nabla V(\omega)$
13:     for $k$ steps do
14:         Sample $\{x^{(j)}\}_{j=1}^{d} \sim p_{data}$ a batch of real data.
15:         Sample $\{z^{(j)}\}_{j=1}^{d} \sim p_{prior}$ a batch of noise.
16:         Get condition $\{y^{(j)}\}_{j=1}^{d}$ from $\{x^{(j)}\}_{j=1}^{d}$
17:         $\{y^{(j)}\}_{j=1}^{d} \leftarrow$ TransLimitOrder($\{y^{(j)}\}_{j=1}^{d}$)
18:         Update $G$ parameter $\theta$ to minimize:
            $V = -\sum_i D(G(z^{(i)} \mid y^{(i)}) \mid y^{(i)})$
            $\theta \leftarrow \theta - \alpha \nabla V(\theta)$
```

The generated limit order can preserve greater diversity than the naive supervised learning model can. The generative model retrieved from a well-trained naive supervised learning model or LOB-GAN is then leveraged as a market behavior simulator to construct Virtual Market. The market behavior simulator is a critical component that allows Virtual Market to synthesize the market's reaction to the agent's trading decision. The generative process and workflow of Virtual Market are introduced in the following subsection.

C. RECONSTRUCTION OF VIRTUAL MARKET

Virtual Market is a general purpose platform that can be leveraged either as a training environment for an RL-based portfolio agent or as a backtest set for investors or a financial model. Furthermore, it can be applied to simulate any type of financial market by leveraging the underlying security matching system and by using historical limit-order data to pretrain the market behavior simulator. In our study, the reconstruction target of Virtual Market was the Taiwan Stock Exchange (TWSE). Therefore, the historical limit-order data for pretraining the market behavior simulator were obtained from the TWSE. In the year of the historical limit-order data utilized in our study, the TWSE implemented a call auction mechanism at a 5 s frequency. Consequently, an identical security matching system was implemented in Virtual Market. When the call auction is executed, the price that can satisfy the maximum volume in an LOB is selected as the strike price. The LOB is a crucial component within Virtual Market that is used to maintain and record the current active limit order on the market. A strike price database is also needed to store the matched strike price on Virtual Market at each time step.

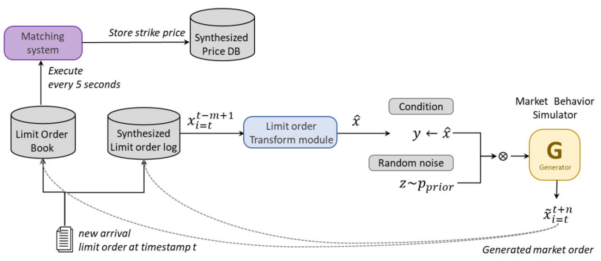


FIGURE 2. Virtual market architecture.

The workflow within each component in Virtual Market is illustrated in Fig. 2. The interaction procedure is described as follows. First, the historical LOB is used to initialize the synthesized LOB for the opening session (8:00 AM to 9:00 AM). Due to a trading rule stipulated by the TWSE, the limit order during the opening session is stored and matched only at the initiation of the first matching. After the initialization phase, the market behavior simulator iteratively retrieves the condition from the synthesized LOB to predict the relative limit-order quantity at the next time step. Regardless of when the agent places an order, the generative process is always executed continuously and sequentially. After the agent places an order, the order merges into the synthesized LOB as part of the condition for the next round of market behavior prediction. The matching system automatically executes security matching on the synthesized LOB every 5 s to determine the strike price that can satisfy the greatest number of order requirements. The strike price at each time step is stored in the synthesized price database and provides a market observation to the agent.

Virtual Market provides a more realistic training environment for RL-based portfolio agents by simulating how the market reacts to the agent's trading decision. Therefore, the agent's relationship with Virtual Market is an interactive one in which one influences the other. In contrast to the unchangeable price fluctuations constructed using historical price data, price fluctuations in Virtual Market are derived from the interplay between the agent and simulated market. Therefore, instead of merely (over)fitting to historical price data, which potentially yields poor generalizations in the testing period, the replaying agent in Virtual Market can apply their trading knowledge to novel contexts during optimization.

D. OTHER GENERALIZATION STRATEGIES

In addition to proposing Virtual Market, we introduce two generalization strategies, borrowing from previous studies on generalization in RL trading. The first strategy is adversarial training, which has been utilized in RL applications to make the agent more robust. Liang *et al.* [9] implemented adversarial training by injecting random noise into price sequences to improve the agent's generalization ability. The authors implemented a similar method in a historical price–based environment during agent training. The price sequence obtained from the training environment at each time step is denoted as X_t. Therefore, adversarial training can be formulated as

$$\begin{aligned} \epsilon &= \mathcal{N}(0, 0.002) \\ X_t &= (1+\epsilon)\cdot X_t, \end{aligned} \tag{4}$$

where ϵ can be treated as a random perturbation on the price sequence and $\mathcal{N}$ is the Gaussian distribution. The injected noise is constrained to 0.2% of the price scale by setting the Gaussian distribution to have a mean of 0 and a standard deviation of 0.002. As is well known, the agent can achieve better generalization ability if it can resist small-scale fluctuations on the input price sequence. The adversarial training makes the training environment more random by adding random perturbation. Another approach to diversifying the training environment is using a $GARCH(p, q)$ model to augment the price sequence. $GARCH(p, q)$ is an autoregressive processe that depends on past variance to predict future variance, where p is the number of lag variances and q is the number of lag residual errors; the model can be described as

$$\begin{aligned} u_t &= \sigma_t \epsilon_t \\ \sigma_t^2 &= \omega + \sum_{i=1}^{p} \alpha_i u_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2, \end{aligned} \tag{5}$$

where $\epsilon_t \overset{iid}{\sim} \mathcal{N}(0, 1),\ \omega > 0,\ \alpha_i \geq 0,\ \beta_j \geq 0$, and $\alpha_i + \beta_j < 1$. The parameter is estimated using maximum likelihood estimation; the GARCH process is detailed in Bollerslev [39]. In this study, the simplest variant $GARCH(1, 1)$ was used to forecast future variance with the purpose of synthesizing a series of price sequences as input. Users of this method can consider injecting perturbation on the price data to increase data diversity.

V. PROPOSED RL-BASED PORTFOLIO FRAMEWORK

Fig. 3 summarizes the proposed RL-based portfolio framework. The framework contains a training environment and agent. Virtual Market instead of historical price data (as used in traditional RL trading) is utilized as a training environment. The agent is an extension of EIIE [7] topology, which is widely discussed in RL-based portfolio research. However, Virtual Market is applied in a learning framework that differs from the original EIIE topology. Thus, the details of policy state space, action space, and reward function are introduced to describe the optimization process.

A. POLICY STATE SPACE DESIGN

The state can be divided into two partitions. The first is the market state (denoted X_t), which is the price sequence obtained from the environment. The second is the self state (denoted U_t), which is used to represent the status of the agent. Because the properties of these two states are heterogeneous, the agent handles these pieces of information individually. The state observed by the agent from Virtual Market at time step t is written as $s_t = (X_t, U_t)$.

The price sequence provided by Virtual Market constitutes a matching result from the security matching system. Therefore, the original time frequency is on a scale of 5 s. To manifest the price fluctuation, the raw strike price within a given period is aggregated every 12 units and converted into an open-high-low-close price (o, h, l, c) format in the minute scale. Within every 12 units of a sequential strike price, the first price represents the open price v^o, the highest price represents the high price v^h, the lowest price represents the low price v^l, and the strike price at the last time step represents the close price v^c. In cooperation with the aforementioned step of price formatting, the market state at each time step X_t comprises a price movement vector $\{V_t^f\}_{f\in\{h,l,c\}}$ in a predefined window size n and formulated as

$$V_t^f = \left\{ \left[\frac{v_{t-n+1}^f}{v_t^f}, \frac{v_{t-n+2}^f}{v_t^f}, \ldots, \frac{v_{t-1}^f}{v_t^f}, \mathbf{1} \right] \right\}_{f\in\{h,l,c\}}, \tag{6}$$

where $v^f \in \{h, l, c\}$ is the high, low, and close price, and $\mathbf{1} = [1, 1, \ldots, 1]^\top$ has a vector length that varies with the amount of assets for a given portfolio. In the Virtual Market setting, we only considered the high, low, and close price movement vector. Thus, the market state can be formulated more precisely as $X_t = \left[V_t^h, V_t^l, V_t^c\right]$.

The self state is designed to make agent decision-making more realistic by having the agent evaluate its own situation. To prevent the agent from changing their portfolio strategy too drastically and too often, the portfolio weight at the previous time step is given as an input observation for the agent. Therefore, the self state can be formulated as $U_t = \mathrm{w}_{t-1}$, where w_t is the weight of each asset within a portfolio that is decided by the agent at the previous time step.

B. CONTINUOUS PORTFOLIO ACTION

At each time step t, the agent takes action a_t, which is synonymous with the portfolio weight w_t. The portfolio action is represented as

$$\begin{aligned} \mathrm{w}_t &= [w_{0,t}, \ldots, w_{q,t}]^\top \\ \text{s.t.} \sum_{i=0}^{q} w_{i,t} &= 1, \quad \forall t, \end{aligned} \tag{7}$$

where $w_{i,t}$ $(1 \leq i \leq q)$ is the weight of the ith asset at time step t, and q is the amount of assets within the portfolio. The first element in the portfolio weight $w_{0,t}$ indicates the weight on cash. The agent always starts trading with $\mathrm{w}_0 = [1, 0, \ldots, 0]^\top$; this means that all cash is initially in the agent's hand. Because the portfolio weight indicates the assignment of capital, it has a constraint that requires the sum of the weights to be 1 at each time step. To fulfill

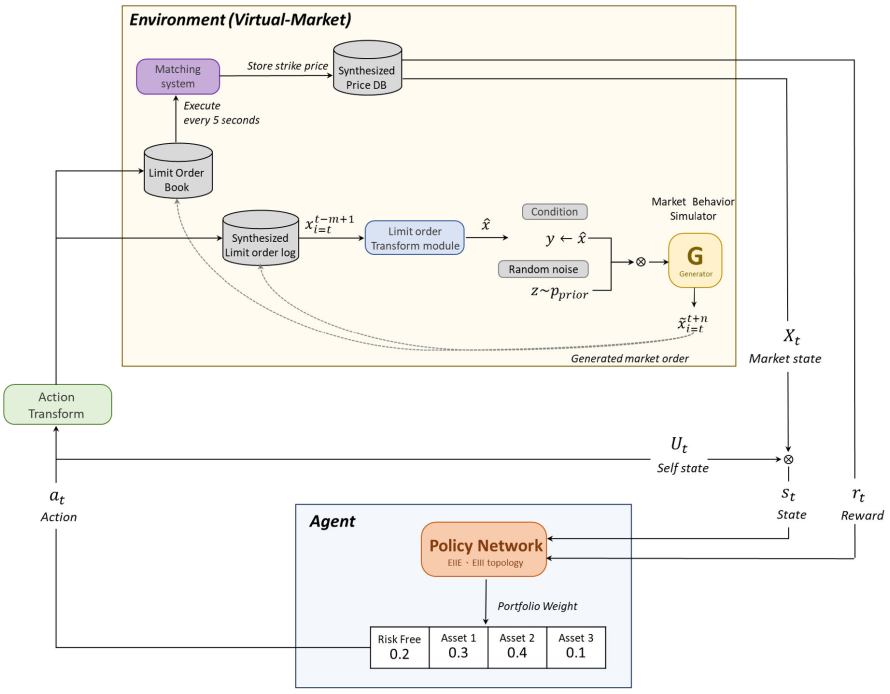


FIGURE 3. Structure of the proposed RL-based portfolio framework.

the hard constraint on the portfolio weight, Softmax normalization is applied. While interacting with Virtual Market, the agent must specify the order quantity and price. Consequently, the portfolio weight at each time step t is transformed into O_t^{agent} based on the portfolio value p_t. The term p_t can be obtained from further inference based on the prior weight w_{t-1} and the price relative vector y_t. The term y_t can be interpreted as the price return that indicates the relative movement of price; if price increases and decreases from $t-1$ to t, then the fraction is greater and less than 1, respectively.

$$\mathrm{y}_t = \frac{v_t^c}{v_{t-1}^c} = \left[1, \frac{v_{1,t}^c}{v_{1,t-1}^c}, \ldots, \frac{v_{q,t}^c}{v_{q,t-1}^c}\right]^\top \tag{8}$$

However, due to the price movement between time steps, the portfolio weight at a previous time step w_{t-1} evolves into w'_t as follows:

$$\mathrm{w}'_t = \frac{\mathrm{y}_t \odot \mathrm{w}_{t-1}}{\sum_{i=1}^{q} \left(\mathrm{y}_{i,t} \cdot \mathrm{w}_{i,t-1}\right)}, \tag{9}$$

where $\odot$ is the element-wise product operation. According to Equation (9), the portfolio value is denoted as $p_t = p_{t-1}\mathrm{y}_t \cdot \mathrm{w}'_t$. Based on p_t, the order placed by agent at time step t is formulated as

$$O_t^{agent} = \left\{ \frac{C \cdot p_t \cdot (\mathrm{w}_{i,t} - \mathrm{w}'_{i,t})}{v_{i,t}^{order}} \right\}_{i=1}^{q} \tag{10}$$

where C is the cash held initially, q is the number of portfolio assets, and $v_{i,t}^{order}$ is the limit-up price and limit-down price for the ith asset at time step t. Therefore, the order placed by the agent is a type of market order. The aforementioned conversion procedure is executed at each time step when the agent interacts with the environment.

C. COST-AWARE REWARD FUNCTION

To evaluate the portfolio switching cost at time step t, the previous weight w_{t-1} must be rebalanced. Thus, at the beginning of time step t, the portfolio weight w'_t is reestimated using Equation 9. After the portfolio weight at the beginning of time step t is retrieved, the portfolio switching remainder factor ψ_t can be formulated as

$$\psi_t = 1 - \left(c \sum_{i=1}^{q} |\mathrm{w}'_{i,t} - \mathrm{w}_t| \right), \tag{11}$$

where c is the transaction cost and q is the number of portfolio assets. The transaction cost is triggered only by changing the holding position. Consequently, the reward r_t can be defined as

$$\begin{aligned} r_t &= \log \frac{p_t}{p_{t-1}} \\ &= \log(\psi_t \mathrm{y}_t \cdot \mathrm{w}_{t-1}). \end{aligned} \tag{12}$$

The final goal of the agent is to maximize the final total reward; in doing so, the agent identifies the long-term portfolio control strategy that achieves the optimal trade-off between revenue and cost.

D. POLICY OPTIMIZATION

Suppose that the policy of the agent (denoted π_ϕ) is parameterized by ϕ. The policy π_ϕ is responsible for determining a portfolio strategy w_t according to observation s_t. Therefore, construction of a portfolio strategy is $a_t = \pi_\phi(s_t)$. The output a_t is considered the optimal action for a given state. The agent is optimized through constant interaction with Virtual Market. Throughout the process, the target of the agent is to maximize the final reward R, which is expressed as $R = \frac{1}{T}\sum_{i=1}^{T} r_t$ where T is the length of an episode. Therefore, the objective function of π_ϕ can be formulated as

$$J(\pi_\phi) = \mathbb{E}\left[p_0 \prod_{t=1}^{T} r(s_t, \pi_\phi(s_t))\right], \tag{13}$$

where p_0 is the initial portfolio value, which is always initialized as 1, and r is the reward function. The objective function highlights the final goal of policy π_ϕ, which is set toward the reward function. Because the goal of a policy is to maximize total reward, the optimal policy is formulated as

$$\begin{aligned}\pi_\phi^* &= \underset{\phi}{\arg\max} \quad \mathbb{E}\left[J(\pi_\phi)\right] \\ &= \underset{\phi}{\arg\max} \quad \mathbb{E}_{\tau\sim\pi_\phi(\tau)}\left[\sum_{t=1}^{T} \log(\psi_t \mathrm{y}_t \cdot \mathrm{w}_{t-1})\right].\end{aligned} \tag{14}$$

The criterion embedded in the formulation of (14) is derived from (12). The optimal policy is obtained based on the inference that the policy always selects the most advantageous action for any state s_t. After the optimization criterion is established, the gradient ascent algorithm is applied for the agent to iteratively update the model parameters by using the following equation:

$$\nabla_\phi J(\pi_\phi) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} R(\tau^n)\nabla_\phi \log \pi_\phi(s_t, a_t), \tag{15}$$

where τ is the number of trajectories, and $R(\tau^n)$ is the total reward for trajectory n. Using the equation, the agent calculates the gradient for each update by sampling N trajectories. The gradient vector is then utilized to update policy parameters ϕ per Equation 16, where η is the learning rate.

$$\phi \leftarrow \phi + \eta\nabla_\phi J(\pi_\phi). \tag{16}$$

The fundamental elements within the RL-based portfolio algorithm are fully introduced. The details of the interaction between each component in Virtual Market and the learning framework of our RL-based portfolio agent are both described in Algorithm 4. From line[4] to line[10], the policy retrieves state s_t from Virtual Market and decides a portfolio weight a_t accordingly. The portfolio weight for each asset is then transformed into an order quantity O_t^{agent} per (10) as the new arrival order made by the agent. Adding the agent order into Virtual Market and using the market behavior simulator G to generate the market order at the next time step is realized at line[11] to line[14]. The matching process is executed over the market current active order $\tilde{B}$ to obtain the strike price. The strike price is then stored back into the price database $\tilde{Q}$. The agent and market iteratively affect each other through this generative process. Furthermore, the agent can obtain a reasonable market reaction based on its trading decision. The agent is optimized by interacting with Virtual Market, as shown in line[17] to line[20]. By using Virtual Market, the agent can influence the state transition in the training environment; such influence cannot be simulated using historical data alone. The proposed learning framework can help the agent apply their trading knowledge to novel contexts and tighten the relationship between the transition state and agent's action.

Algorithm 4 Proposed RL-Based Portfolio Framework

```
Require: Learning rate $\eta$, asset number $q$, epoch $M$, trajectory length $T$, update frequency $N$, transition buffer $D$
Require: Initial policy $\pi$ parameter $\phi$
Require: Pretrained LOB-GAN generator $G$
Require: Stock matching system $Match$, limit-order transform module $Trans$
Require: Initial Virtual Market LOB $\tilde{B}$, order record $\tilde{O}$, price database $\tilde{Q}$ with historical data
1:  Initial action $a_0 = \{\frac{1}{q}, \frac{1}{q}, \dots, \frac{1}{q}\}$
2:  for $epoch = 1, \dots, M$ do
3:      for $t = 1, \dots, T$ do
4:          Get market state $X_t$ from $\tilde{Q}_t$
5:          Observe state $s_t = (X_t, a_{t-1})$
6:          Select action $a_t = \pi_\phi(s_t)$ based on current policy
7:          Transform $a_t$ into agent order $O_t^{agent}$
8:          Merge $O_t^{agent}$ into $\tilde{B}$ and update to $\tilde{O}_t$
9:          Get strike price $v_{t+1} \leftarrow Match(\tilde{B})$
10:         Format price $\{V_{t+1}^f\}_{f\in\{h,l,c\}} \leftarrow OHLC(v_{t+1})$
11:         Store $\{V_{t+1}^f\}_{f\in\{h,l,c\}}$ back to $\tilde{Q}_{t+1}$
12:         Sample $z$ from $\mathcal{N}(0,1)$ and $y_t$ from $\tilde{O}_t$
13:         Predict $O_{t+1}^{market} \leftarrow G(z \mid Trans(y_t))$
14:         Merge $O_{t+1}^{market}$ into $\tilde{B}$ and update to $\tilde{O}_{t+1}$
15:         Observe reward $r_t$
16:         Store transition $(s_t, a_t, r_t)$ in $D$
17:     if $epoch \mod N = 0$ then
18:         Calculate $\nabla_\phi J(\pi_\phi)$ by Equation 15
19:         Update policy: $\phi \leftarrow \phi + \eta\nabla_\phi J(\pi_\phi)$
20:         Empty transition buffer $D$
```

VI. EXPERIMENTAL RESULTS

Three experiments were conducted. The first demonstrated that Virtual Market improved the optimization of the RL-based portfolio agent, the second demonstrated that our generalization strategy outperforms its counterparts, and the third demonstrated that our strategy yields optimal portfolio performance among its counterparts. The subsequent subsections first describe the experimental setting before describing each experiment in turn.

TABLE 2. Hyperparameters of the proposed framework.

Model	Hyperparameter	Value
LOB-GAN	Learning rate	0.0002
	Window size	12
	Generate size	1
RL framework	Learning rate	0.002
	Number of assets	11
	Decision period (min)	30
	Window size	50
	Generative step number	5760
	Transaction cost	0.25%
	Initial cash	1×10^7

TABLE 3. Portfolio assets within each portfolio combination.

Portfolio combination	port#1	port#2	port#3
Portfolio assets	00632R.TW	2317.TW	2331.TW
	00633L.TW	2474.TW	6235.TW
	2303.TW	2354.TW	2368.TW
	2330.TW	2312.TW	6191.TW
	2409.TW	2464.TW	3016.TW
	2448.TW	6139.TW	8110.TW
	3481.TW	3450.TW	4934.TW
	2881.TW	3665.TW	8105.TW
	2882.TW	2488.TW	3380.TW
	2884.TW	2390.TW	4118.TW
	2891.TW	2404.TW	8404.TW

A. EXPERIMENTAL SETTING

In our experiments, we used minute-level price data and limit-order-log data from the TWSE for the period of January to December 2016. The minute-level price data were used to construct a benchmark environment against which Virtual Market was compared. An environment constructed based on historical data was used as a static, noninteractive simulator for the agent, and the limit-order-log data were used to pretrain our proposed market behavior simulator.

The limit-order data must go through a two-phase procedure. In the first phase, *data aggregation* is executed by aggregating limit-order quantities with the same order price every 5 s. The aggregated order quantity can be considered an ordering expectation from overall investors. In the second phase, *data reduction* is applied to retain orders having a ± 10 price difference relative to the strike price at the previous time step; doing so can not only appreciably reduce noise considerably within the limit order but also help the model learn a market ordering behavior that has greater generality. The limit orders obtained after this two-phase procedure were used to pretrain the market behavior simulator.

Because the price fluctuation of each asset varied within time periods, a rolling test was applied in the experiments aimed at verifying portfolio performance. Each rolling test set comprised data from 3 months of training and 1 month of testing. For example, the market behavior simulator was trained on limit-order data for the January–March 2016 period, and the well-trained generative model was used to construct Virtual Market as the training environment for the agent. Agent performance was then evaluated using April 2016 data. For the subsequent round of rolling tests, the training and testing periods shifted forward by one month. Under this rolling test setup, the data utilized in our experiment can be divided into nine partitions to cross-check the generalization ability of each portfolio strategy.

Our LOB-GAN use 2 convolutional and then 5 fully-connected layers as generator; for the discriminator, we use 3 convolutional and 4 fully-connected layers.

The hyperparameter settings of our learning framework are presented in Table 2. The LOB-GAN was set to use the previous 12 time steps of the limit order as the condition to generate the relative order quantity distribution for the subsequent time step. For the RL trading framework, the agent was set to simultaneously manipulate 11 assets and incur a 0.25% transaction cost when switching between portfolio strategies. The trading frequency is defined as the time interval between two portfolio decisions. The cash held initially is used to represent the capital assigned to the agent. An agent with more capital can place an order for a greater quantity, which perturbs the market more greatly. The window size represents the length of the price sequence obtained by the agent from the environment at each time step. The generative step number indicates the generative process between the agent and Virtual Market within an episode.

In our study, the agent was set to be responsible for only portfolio weighting. Therefore, we applied 3 distinct heuristic asset selection rules to select 11 assets from the TWSE. The assets of each portfolio combination are listed in Table 3. The selection rule for each portfolio combination was as follows:

- Considering that volume is a critical metric for evaluating market liquidity, port #1 was constructed by selecting the 11 highest-volume assets on the 2016 TWSE.
- Port #2 first restricts the asset pool within other firms in the electronics industry and then selects the 11 highest-volume assets within the asset pool.
- Port #3 first ranks all assets by market capitalization and retains mid-cap and small-cap assets. Second, to better simulate the agent's impact on Virtual Market, port #3 filters the asset pool by using the criterion that >30% of stocks are held by directors and supervisors. Third, the 11 highest-volume assets within the remaining asset pool were selected to ensure liquidity.

Three indicators were used to evaluate the performance of a portfolio strategy. The first indicator was the accumulative portfolio value (APV), which is defined as

$$APV = p_0 \prod_{t=1}^{T} p_t, \tag{17}$$

where p_0 is the initial portfolio value, and T is the time span of a given portfolio management process. A higher APV indicates greater profitability in portfolio management. The second indicator was maximum drawdown (MDD), which is defined as the maximum loss from a peak to a trough of a portfolio return as follows:

$$MDD = \max_{\tau \in (0,T)} \left[\max_{t \in (0,\tau)} \left[\frac{p_t - p_\tau}{p_t} \right]_+ \right]. \tag{18}$$

A lower *MDD* indicates more stable returns from a portfolio strategy. The third indicator was the Sharpe ratio (SR), which encapsulates both profit and risk. The *SR* is defined as the average portfolio return in excess of a risk-free rate under one unit of risk:

$$SR = \frac{\mathbb{E}\left[\rho_t - \rho_f\right]}{\sqrt{var\left(\rho_t - \rho_f\right)}}, \tag{19}$$

where $\rho_t = \frac{p_t}{p_{t-1}}$ is the portfolio return, and ρ_f is the risk-free asset rate. A higher *SR* indicates a portfolio's greater profitability per unit risk.

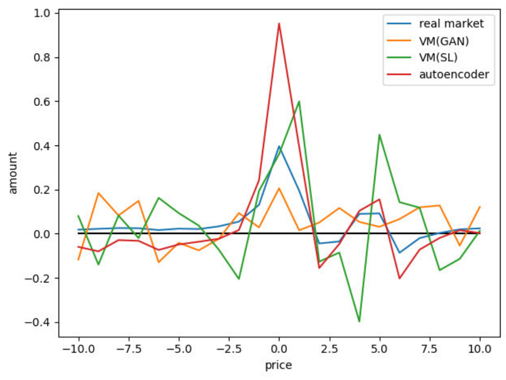
Although we use these three evaluation metrics, APV is prioritized. Because our primary goal is portfolio management, our aim is to maximize the size and stability of profits. MDD and SR are metrics that are only used for evaluating the results.

B. IMPROVEMENT DUE TO VIRTUAL MARKET

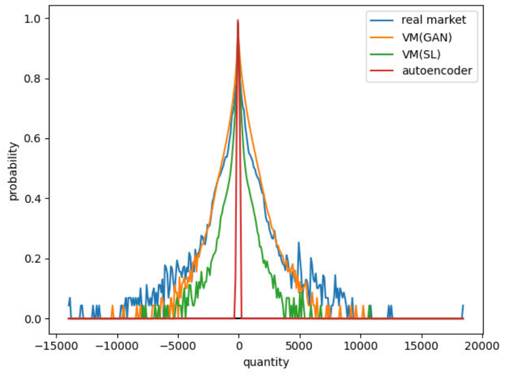
Before we improve the virtual market, we must evaluate how well the virtual market performs. We compare our VM (GAN) and VM (SL) with the autoencoder and with a real market. The results are shown in Fig. 4. Specifically, Fig. 4(a) presents the price–quantity distribution, which describes how many stocks are ordered by investors at a given price. The unit for price is the price level, where 0.0 represents the current strike price. As evident in Fig. 4(a), the VM (GAN) outperforms the VM (SL) and the autoencoder; it generates a curve with no overly large peaks or troughs that are closest to the real market curve. The strong performance of the VM (GAN) is further evidenced in Fig. 4(b). The figure illustrates the quantity generated by the generative models. Both VM (GAN) and VM (SL) outperform the autoencoder, as indicated by the near-zero value that it generated. A likely explanation for the autoencoder's poor performance is the presence of many zeros in the training data, which makes data difficult to learn. Furthermore, VM (GAN) and VM (SL) perform similarly, but VM (GAN) is good at generating occasional outliers, which are present in a real market.

To verify Virtual Market's ability to yield improvements in the optimization of various types of RL-based portfolio agents, in this experiment, Virtual Market was applied to the four following policy networks:

- *CNN EIIE (CNN) [7]:* A convolutional neural network (CNN) was used to extract a fixed size of the price movement trend. The feature within each asset is processed individually.
- *Vanilla RNN EIIE (RNN) [7]:* A vanilla recurrent neural network (RNN) cell was applied to extract features on the historical market price for each portfolio asset.
- *LSTM EIIE (LSTM) [7]:* A time series feature extractor network was switched to a long short-term memory (LSTM) cell to better capture time-series information.



(a) Price distribution



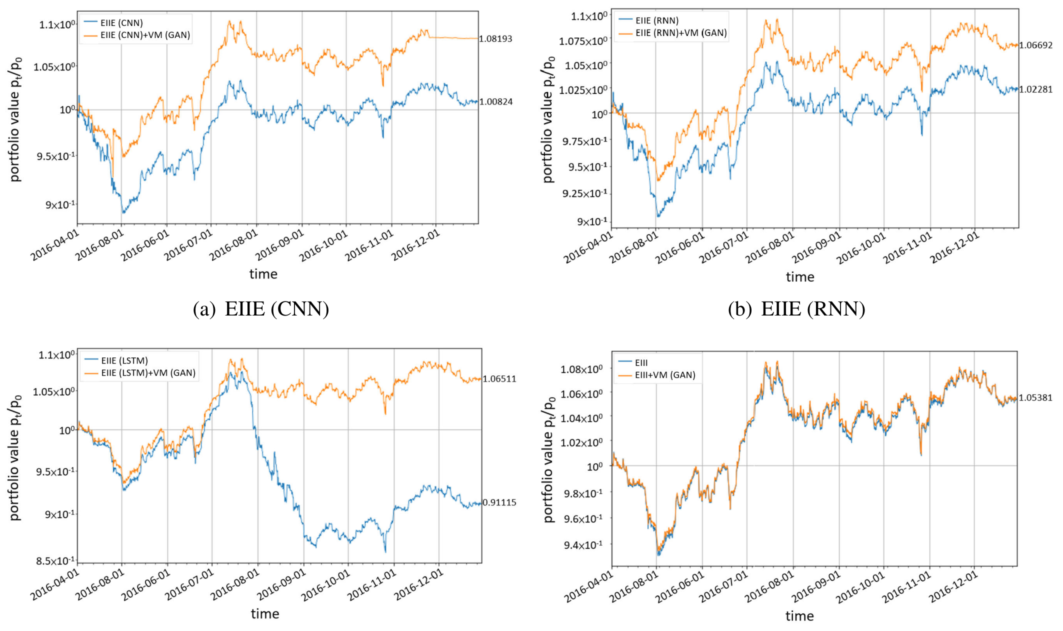
(b) Quantity distribution

FIGURE 4. Comparison of virtual market to real market.

- *EIII [5]:* An extension of the EIIE topology, the inception network was leveraged to extract multi-scale information on time-series data.

The experimental results are presented in Table 4. VM (GAN) stands for Virtual Market as constructed by LOB-GAN. For the *MDD* evaluation, EIIE (RNN) + VM (GAN) under port #1 had an *MDD* of 0.02905, which was a slight deprovement. EIII + VM (GAN) under port #3 had an *MDD* 0.05307, which was also a slight deprovement. By contrast, the agent optimized using Virtual Market performed better most of the time. For *APV*, Virtual Market yielded significant improvements. Only for EIII + VM (GAN) under port #3 was there a slight deprovement from the original EIII (an *APV* of 0.97060 vs. 0.98516). For *SR*, Virtual Market's performance exhibited greater variance relative to the other evaluation metrics. The agent optimized using VM (GAN) did not outperform the original agent in several cases. However, VM (GAN) yielded a more stable overall portfolio performance.

The overall portfolio improvements due to VM (GAN) were more significant on port #2 and port #3 than they were on port #1. Upon analyzing the property of each portfolio combination, we noticed that stocks in port #1 were mostly large-cap stocks, suggesting that the agent had less market impact on port #1 relative to the other combinations. The experimental results indicated that our proposed



(a) EIIE (CNN) (b) EIIE (RNN)

(c) EIIE (LSTM) (d) EIII

FIGURE 5. Improvement in portfolio value due to virtual market.

TABLE 4. Generalization improvements of policy networks.

Portfolio	Model	Evaluation metric		
		MDD	*APV*	*SR*
port#1	EIIE (CNN)	0.04138	0.98756	-0.02747
	EIIE (RNN)	**0.02823**	1.00408	**0.02186**
	EIIE (LSTM)	0.04419	0.99166	-0.00547
	EIII	0.02999	1.00608	**0.01326**
	EIIE (CNN)+VM (GAN)	**0.02997**	**1.00566**	**0.01035**
	EIIE (RNN)+VM (GAN)	0.02905	**1.00753**	0.01321
	EIIE (LSTM)+VM (GAN)	**0.02926**	**1.00735**	**0.01327**
	EIII+VM (GAN)	**0.02941**	**1.00614**	0.01114
port#2	EIIE (CNN)	0.07265	0.97919	-0.01698
	EIIE (RNN)	**0.03540**	1.00110	-0.00198
	EIIE (LSTM)	0.05906	0.97916	-0.01502
	EIII	0.04785	0.98852	-0.08030
	EIIE (CNN)+VM (GAN)	**0.03287**	**1.00538**	**0.00954**
	EIIE (RNN)+VM (GAN)	0.03814	**1.00448**	**0.00678**
	EIIE (LSTM)+VM (GAN)	**0.03500**	**1.00224**	**-0.00064**
	EIII+VM (GAN)	**0.03410**	**1.00117**	**-0.00129**
port#3	EIIE (CNN)	0.03472	0.99304	-0.01855
	EIIE (RNN)	0.03421	1.00412	0.00536
	EIIE (LSTM)	0.05602	0.97262	-0.02690
	EIII	**0.04655**	**0.98516**	-0.01854
	EIIE (CNN)+VM (GAN)	**0.02963**	**1.00554**	**0.02395**
	EIIE (RNN)+VM (GAN)	**0.02534**	**1.00812**	**0.01453**
	EIIE (LSTM)+VM (GAN)	**0.02870**	**0.99773**	**-0.00080**
	EIII+VM (GAN)	0.05307	0.97060	**0.01050**

VM (GAN) yields lower reliability for the portfolio combinations in which the agent has less market impact. However, our VM (GAN) could still achieve reliable testing performance under different portfolio combinations. We also noted that our VM (GAN) was less sufficient on the EIII topology than it was on other policy networks. However, as evident in the table, improvements are still present in port #1 and the EIII topology. This finding indicates that our VM (GAN) still greatly improves the overall portfolio performance on average.

Portfolio performance is compared between the use of Virtual Market–generated data and the use of real market data (Fig. 5). Because portfolio management must be based on some strategy, we select four distinct strategies for the given task in our comparison. Each strategy is applied to Virtual Market–generated data and compared against its identical counterpart as applied to real market data.

In Fig. 5, it illustrates the accumulated portfolio value of the four policy networks. In the figure, the blue line represents the agent optimized using a historical price–based environment, and the orange line represents the agent optimized using VM (GAN). The use of VM (GAN)—in conjunction with EIIE (CNN), EIIE (RNN), and EIIE (LSTM)—clearly yielded a greater accumulated portfolio value than the use of the historical price–based environment did over the testing period. However, the improvement in portfolio performance due to VM (GAN) was less significant for the EIII policy network than it was for the other policy networks. The original EIIE (CNN), EIIE (RNN), and EIIE (LSTM) obtained portfolio values of 1.08193, 1.02281, and 0.91115, respectively. By contrast, the EIIE (CNN) + VM (GAN), EIIE (RNN) + VM (GAN), and EIIE (LSTM) + VM (GAN) obtained portfolio values of 1.08193, 1.06692, and 1.06511, respectively. The application of the VM (GAN) to the EIII network improved the accumulated portfolio value from 1.05299 to 1.05381. Therefore, using our LOB-GAN to construct Virtual Market as a training environment can improve the generalization ability of various types of portfolio agents.

TABLE 5. Comparison with different generalization methodologies.

Portfolio	Method	Evaluation metric		
		MDD	APV	SR
port#1	EIIE (RNN)	0.02823	1.00408	**0.02186**
	EIIE (RNN)+VM (GAN)	0.02905	**1.00753**	0.01321
	EIIE (RNN)+VM (SL)	0.03033	1.00671	0.01310
	EIIE (RNN)+Adv.	**0.02227**	0.99887	0.01329
	EIIE (RNN)+GARCH	0.02371	0.99365	-0.00020
port#2	EIIE (RNN)	0.03540	1.00110	0.00198
	EIIE (RNN)+VM (GAN)	0.03814	**1.00448**	**0.00678**
	EIIE (RNN)+VM (SL)	0.03559	1.00175	-0.00061
	EIIE (RNN)+Adv.	**0.03533**	1.00190	-0.00040
	EIIE (RNN)+GARCH	0.03676	0.98257	-0.04657
port#3	EIIE (RNN)	0.03421	1.00412	0.00536
	EIIE (RNN)+VM (GAN)	**0.02534**	**1.00812**	0.01453
	EIIE (RNN)+VM (SL)	0.03431	1.00399	**0.05569**
	EIIE (RNN)+Adv.	0.03421	1.00408	0.00527
	EIIE (RNN)+GARCH	0.02681	1.00121	0.00206
Avg.	EIIE (RNN)	0.03261	1.00310	0.00973
	EIIE (RNN)+VM (GAN)	0.03084	**1.00671**	0.01151
	EIIE (RNN)+VM (SL)	0.03341	1.00415	**0.02273**
	EIIE (RNN)+Adv.	0.03060	1.00162	0.00605
	EIIE (RNN)+GARCH	**0.02915**	0.99248	-0.01490

C. COMPARISON OF GENERALIZATION STRATEGIES

In this experiment, the proposed VM (GAN) was compared against the other generalization strategies introduced in Section IV. The experimental results are presented in Table 5. For simplification, VM (SL) stands for Virtual Market with a naive supervised learning market behavior model, Adv. stands for a historical price–based environment with adversarial training, and GARCH stands for an environment utilizing GARCH(1,1) to synthesize the price sequence. Under port #1, *MDD* performance improved significantly from the use of adversarial training and GARCH (0.02227 and 0.02371, respectively, on average). Virtual Market was less effective; VM (GAN) and VM (SL) obtained 0.02905 and 0.03033, respectively. With respect to *APV*, Virtual Market outperformed other generalization strategies. The VM (GAN) yielded the greatest profit at $APV = 1.00752$. With respect to *SR*, the original setting performed best at $SR = 0.02186$. The original EIIE (RNN) achieved comparable profitability at less volatility, indicating that VM (GAN) achieves the greatest profit at the expense of greater downward volatility (which was the highest among strategies). Adversarial learning yielded the most stable portfolio but at a considerable expense to profitability. For the port #2 combination, adversarial training yielded the best *MDD* at 0.03533; the proposed VM (GAN) and VM (SL) yielded *MDD* values of 0.03814 and 0.03559, respectively; however, the performance was still worse than that of the original EIIE (RNN). VM (GAN) yielded the best *APV* and *SR* at 1.00448 and 0.00678, respectively. Under port #3, Virtual Market yielded the best improvements to generalization ability. VM (GAN) had the best *MDD* and *APV* of 0.02534 and 1.00812, respectively, and VM (SL) had the best *SR* at 0.05569—although VM (GAN) performed second best at $SR = 0.01453$. As noted in the row of average values, although the adversarial training and GARCH techniques exhibit the best performance in terms of MDD, they exhibit the worst performance in terms of APV and SR, even when compared with their original counterparts. Nevertheless, VM (GAN) and VM (SL) perform comparably in terms of APV and SR, and VM (GAN) performs almost as well as adversarial training in terms of MDD. The experimental results indicated that Virtual Market achieved the greatest improvements to generalization ability relative to other strategies. Furthermore, the use of LOB-GAN to construct Virtual Market further resulted in greater stability in generalization than the use of naive supervised learning did. On port #3—which could manifest the advantage of the Virtual Market framework—VM (GAN) yielded better improvements to generalization relative to its counterparts. Notably, Virtual Market yielded better profitability than adversarial training did, although adversarial training yielded the greatest improvement in portfolio stability among the generalization strategies.

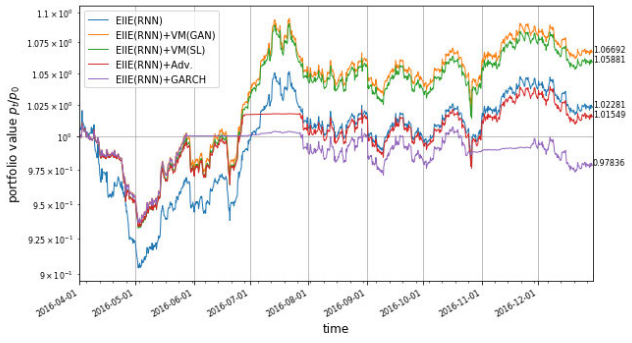


FIGURE 6. Portfolio value curves for various generalization strategies.

Fig. 6 illustrates the accumulated portfolio over all testing periods. The accumulated portfolio values of VM (GAN) and VM (SL) were 1.06691 and 1.05881, respectively. Both VM (GAN) and VM (SL) yielded markedly higher portfolio performance than traditional RL did. The adversarial training and GARCH methods had the ultimate portfolio values of 1.01549 and 0.97836, respectively. Adversarial training and GARCH yielded much lower portfolio volatilities than their counterparts did. As indicated in the figure, the strategy that the agents adopted between July and August 2016, which was optimized with adversarial training and GARCH, involved keeping cash on hand most of the time. This strategy resulted in a more stable portfolio value. Between the original EIIE (RNN) and the EIIE (RNN) + VM (GAN), portfolio performance over the testing period increased by 4% as a result of leveraging Virtual Market in RL-based portfolio policy training. Overall, these results indicated that the Virtual Market framework outperformed other generalization techniques on various portfolio combinations. Under different settings of the market behavior simulator, Virtual Market may yield different results. Nonetheless, using LOB-GAN to construct Virtual Market yields a strategy that balances profitability with stability.

D. EVALUATION OF PORTFOLIO PERFORMANCE

This experimental test of portfolio performance compared the agents optimized with Virtual Market with those adopting the following traditional portfolio strategies:

- *Momentum Strategy:* Also known as "follow the winner," this strategy continually increases the weight of profitable assets; this strategy has been applied in the universal portfolios (UP) [1], exponential gradient (EG) [40], and online Newton step (ONS) [41] methods, which we tested.
- *Contrarian Strategy:* Also known as "follow the loser," this strategy is a mean reversion strategy based on the assumption that less profitable assets will subsequently become profitable; this strategy has been applied in the ANTICOR [42], passive aggressive mean reversion (PAMR) [43], online moving average reversion (OLMAR) [44], weighted moving average mean reversion (WMAMR) [45], and robust median reversion (RMR) [2] methods, which we tested.
- *Pattern Matching:* This strategy is based on the assumption that historical patterns recur, and a portfolio is thus constructed based on similar historical sequences. This strategy has been applied in the M0 [46] and correlation-driven nonparametric learning (CORN) [3] methods, which we tested.

Table 6 summarizes the portfolio performance of traditional portfolio strategies and of RL-based agents under different generalization strategies. The following results are those for port #1: EIIE (RNN) + VM (GAN) achieved the best *APV* at 1.00753; among traditional strategies, EG achieved the best *APV* at 1.00379. EIIE (RNN) + Adv. achieved the best *MDD* of 0.02227; among traditional strategies, UP achieved the best *MDD* of 0.02801. EG (a traditional strategy) had the best *SR* at 0.00496, which was higher than the *SR* of 0.01321 of EIIE (RNN) + VM (GAN).

The following results are those for port #2, UP yielded the best portfolio performance among traditional strategies at the *MDD*, *APV*, and *SR* of 0.03951, 1.00430, and 0.00811, respectively. As for RL-based strategies, EIIE (RNN) + Adv. yielded the best *MDD* at 0.03533, and EIIE (RNN) + VM (GAN) yielded the best *APV* and *SR*. EIIE (RNN) + VM (GAN) yielded a better *MDD* and *APV* than UP did. Nonetheless, EIIE (RNN) + VM (GAN) and UP yielded similar portfolio performance. In the experiment, EIIE (RNN) + VM (GAN) tended to learn a portfolio strategy that was akin to the momentum strategy, but at lower portfolio volatility and higher profitability relative to the traditional momentum strategy. For the port #3 combination, UP was the most stable and profitable among traditional strategies with an *MDD*, *APV*, and *SR* of 0.03921, 1.00769, and 0.01567, respectively. The best RL-based strategies were those in which Virtual Market was used: EIIE (RNN) + VM (GAN) outperformed UP with an *MDD* of 0.02534 and an *APV* of 1.00812, and EIIE (RNN) + VM (SL) had the highest *SR* of 0.05569; EIIE (RNN) + VM (GAN) had a comparable *SR* of 0.01453.

TABLE 6. Comparison with various traditional portfolio approaches.

Portfolio	Model	Evaluation Metric		
		MDD	APV	SR
port#1	EIIE (RNN)	0.02823	1.00408	**0.02186**
	EIIE (RNN)+VM (GAN)	0.02905	**1.00753**	0.01321
	EIIE (RNN)+VM (SL)	0.03033	1.00671	0.01310
	EIIE (RNN)+Adv.	**0.02227**	0.99887	0.01329
	EIIE (RNN)+GARCH	0.02371	0.99365	-0.00020
	UP	**0.02801**	1.00377	0.00484
	EG	0.02806	**1.00379**	**0.00496**
	ONS	0.03165	0.99696	-0.00693
	Anticor	**0.06735**	**0.93885**	**-0.17672**
	PAMR	0.34797	0.65284	-0.61840
	OLMAR	0.27202	0.73681	-0.38119
	WMAMR	0.15974	0.84478	-0.24531
	RMR	0.27098	0.73632	-0.38345
	M0	**0.03582**	**0.99136**	**-0.03875**
	CORN	0.29520	0.70459	-0.67213
port#2	EIIE (RNN)	0.03540	1.00110	0.00198
	EIIE (RNN)+VM (GAN)	0.03814	**1.00448**	**0.00678**
	EIIE (RNN)+VM (SL)	0.03559	1.00175	-0.00061
	EIIE (RNN)+Adv.	**0.03533**	1.00190	-0.00040
	EIIE (RNN)+GARCH	0.03676	0.98257	-0.04657
	UP	**0.03951**	**1.00430**	**0.00811**
	EG	**0.03951**	1.00420	0.00784
	ONS	0.04079	0.99853	-0.00719
	Anticor	**0.07094**	**0.95028**	**-0.09935**
	PAMR	0.35497	0.64560	-0.49155
	OLMAR	0.27832	0.73070	-0.32877
	WMAMR	0.16464	0.84764	-0.20390
	RMR	0.28619	0.72152	-0.34056
	M0	**0.05858**	**0.98884**	**-0.01949**
	CORN	0.32911	0.68317	-0.45996
port#3	EIIE (RNN)	0.03421	1.00412	0.00536
	EIIE (RNN)+VM (GAN)	**0.02534**	**1.00812**	0.01453
	EIIE (RNN)+VM (SL)	0.03431	1.00399	**0.05569**
	EIIE (RNN)+Adv.	0.03421	1.00408	0.00527
	EIIE (RNN)+GARCH	0.02681	1.00121	0.00206
	UP	**0.03921**	**1.00769**	**0.01567**
	EG	0.03926	1.00768	**0.01567**
	ONS	0.04342	1.00199	0.00499
	Anticor	**0.06937**	**0.95640**	**-0.08325**
	PAMR	0.32983	0.67599	-0.42022
	OLMAR	0.26796	0.74445	-0.27054
	WMAMR	0.18531	0.82508	-0.19677
	RMR	0.26528	0.74615	-0.26992
	M0	**0.04755**	**1.00063**	**-0.00196**
	CORN	0.34337	0.66072	-0.50118

These experimental results indicate the following. First, among traditional strategies, the contrarian strategy is the least effective, and the momentum strategy, as applied in UP, is the most stable and profitable. Second, EIIE (RNN) + VM (GAN) and UP employ qualitatively similar strategies, but EIIE (RNN) + VM (GAN) outperforms UP in most cases. In general, these results demonstrate that Virtual Market outperforms not only other generalization strategies but also other traditional portfolio strategies.

Fig. 7 presents the portfolio curve of the various portfolio strategies. The most profitable strategies in each strategy category were compared. UP had the most profitable momentum strategy at an accumulated portfolio value of 1.03281, ANTICOR had the highest-performing contrarian strategy

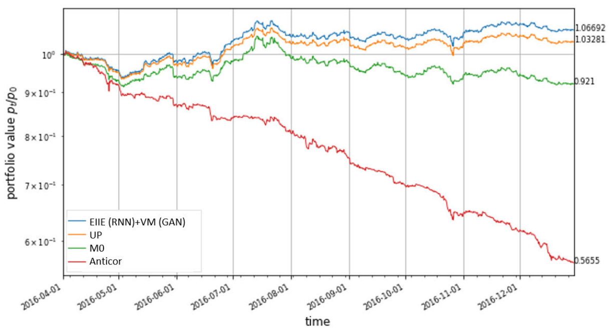


FIGURE 7. Portfolio value curves for various portfolio strategies (cumulative over all testing periods).

at an accumulated portfolio value of (only) 0.5655, and M0 had the highest-performing pattern-matching approach at an accumulated portfolio value of 0.921. Among all portfolio strategies, EIIE (RNN) + VM (GAN) had the highest accumulated portfolio value of 1.06692, and the contrarian strategy yielded the lowest profit (because the market was consistently trending upward during the experimental period). UP and EIIE (CNN) + VM (GAN) had qualitatively similar portfolio value curves, but the EIIE (RNN) + VM (GAN) had a higher portfolio value.

In summary, the experimental results indicate that 1) Virtual Market affords the agent a realistic training environment, 2) optimization using the Virtual Market framework improves the generalization ability of the agent, and 3) Virtual Market outperforms traditional portfolio strategies in portfolio performance.

VII. CONCLUSION AND FUTURE RESEARCH

We proposed a generative model that simulates the ordering behavior of investors in the market. This model was then used to construct a simulated stock exchange, called Virtual Market, as a training environment for an RL-based portfolio agent. We then formulated a novel RL-based portfolio optimization framework that utilizes Virtual Market to improve the generalization ability of the trading agent. Our contributions include the following:

- Our LOB-GAN models the distribution that underlies ordering behavior in the market.
- By utilizing a limit-order transform module, the LOB-GAN aims to generate the relative order quantity distribution instead of precisely generating the order price and quantity.
- Virtual Market is constructed by having the generative model of a well-trained market behavior simulator cooperate with a security matching system.
- When Virtual Market is used as a training environment for an RL-based portfolio agent, the agent can obtain realistic market feedback in response to its previous action.

Our experimental results demonstrated that our framework operates reliably under different policy networks to improve the generalization ability of an RL-based portfolio agent. Our framework also yields greater improvements to generalization ability relative to other generalization methodologies. Virtual Market can also prevent the agent from overfitting to the historical environment during the exploration and trial-and-error process; Virtual Market does so by amplifying the feedback given by the environment in response to the agent's previous action. Virtual Market yielded an approximately 4% improvement to testing portfolio performance.

Future studies can address several issues. First, the diversity of the generated limit order remains unverified, and an evaluation metric is required. Second, the Virtual Market can be further extended to a benchmarking metric or an environment as a means to evaluate generalization ability as it relates to portfolio strategy. Finally, Virtual Market's ability to handle complexity requires further examination; its ability to reconstruct complex trading systems and the challenge of applying the generative model to capture complex market ordering behavior are topics that warrant further investigation.

REFERENCES

[1] T. M. Cover, "Universal portfolios," in *The Kelly Capital Growth Investment Criterion: Theory and Practice*. Singapore: World Scientific, 2011, pp. 181–209.

[2] D.-J. Huang, J. Zhou, B. Li, S. C. H. Hoi, and S. Zhou, "Robust median reversion strategy for online portfolio selection," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 9, pp. 2480–2493, Sep. 2016.

[3] B. Li, S. C. H. Hoi, and V. Gopalkrishnan, "CORN: Correlation-driven nonparametric learning approach for portfolio selection," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–29, Apr. 2011.

[4] J. Wang, Y. Zhang, K. Tang, J. Wu, and Z. Xiong, "AlphaStock: A buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1900–1908.

[5] S. Shi, J. Li, G. Li, and P. Pan, "A multi-scale temporal feature aggregation convolutional neural network for portfolio management," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, Nov. 2019, pp. 1613–1622.

[6] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.

[7] Z. Jiang, D. Xu, and J. Liang, "A deep reinforcement learning framework for the financial portfolio management problem," 2017, *arXiv:1706.10059*. [Online]. Available: http://arxiv.org/abs/1706.10059

[8] Y. Ye, H. Pei, B. Wang, P.-Y. Chen, Y. Zhu, J. Xiao, and B. Li, "Reinforcement-learning based portfolio management with augmented asset movement prediction states," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 1, 2020, pp. 1112–1119.

[9] Z. Liang, H. Chen, J. Zhu, K. Jiang, and Y. Li, "Adversarial deep reinforcement learning in portfolio management," 2018, *arXiv:1808.09940*. [Online]. Available: http://arxiv.org/abs/1808.09940

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018.

[11] T. Spooner and R. Savani, "Robust market making via adversarial reinforcement learning," 2020, *arXiv:2003.01820*. [Online]. Available: http://arxiv.org/abs/2003.01820

[12] E. F. Sbruzzi, M. C. R. Leles, and C. L. Nascimento, "Introducing learning automata to financial portfolio components selection," in *Proc. Annu. IEEE Int. Syst. Conf. (SysCon)*, Apr. 2018, pp. 1–6.

[13] C. Tang, W. Zhu, and X. Yu, "Deep hierarchical strategy model for multi-source driven quantitative investment," *IEEE Access*, vol. 7, pp. 79331–79336, 2019.

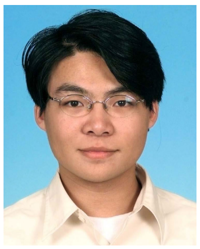
[14] Y. Li, W. Zheng, and Z. Zheng, "Deep robust reinforcement learning for practical algorithmic trading," *IEEE Access*, vol. 7, pp. 108014–108022, 2019.

[15] S. Whiteson, B. Tanner, M. E. Taylor, and P. Stone, "Protecting against evaluation overfitting in empirical reinforcement learning," in *Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn. (ADPRL)*, Apr. 2011, pp. 120–127.
[16] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," 2018, *arXiv:1812.02341*. [Online]. Available: http://arxiv.org/abs/1812.02341
[17] M. Igl, K. Ciosek, Y. Li, S. Tichiatschek, C. Zhang, S. Devlin, and K. Hofmann, "Generalization in reinforcement learning with selective noise injection and information bottleneck," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 13956–13968.
[18] K. Lee, K. Lee, J. Shin, and H. Lee, "Network randomization: A simple technique for generalization in deep reinforcement learning," in *Proc. 8th Int. Conf. Learn. Represent.*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–5.
[19] A. Srinivas, M. Laskin, and P. Abbeel, "CURL: Contrastive unsupervised representations for reinforcement learning," 2020, *arXiv:2004.04136*. [Online]. Available: http://arxiv.org/abs/2004.04136
[20] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner, "DARLA: Improving zero-shot transfer in reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 1480–1490.
[21] S. Gamrian and Y. Goldberg, "Transfer learning for related reinforcement learning tasks via image-to-image translation," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2063–2072.
[22] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 1126–1135.
[23] J. X Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," 2016, *arXiv:1611.05763*. [Online]. Available: http://arxiv.org/abs/1611.05763
[24] B. G. Malkiel, "The efficient market hypothesis and its critics," *J. Econ. Perspect.*, vol. 17, no. 1, pp. 59–82, Feb. 2003.
[25] D. S. Scharfstein and J. C. Stein, "Herd behavior and investment," *Amer. Econ. Rev.*, vol. 80, pp. 465–479, Jun. 1990.
[26] M. Lovric, U. Kaymak, and J. Spronk, *A Conceptual Model of Investor Behavior*, document ERIM Rep. Ser. Reference No. ERS-2008-030-F&A, 2008.
[27] K. V. A. Shantha, C. Xiaofang, and L. P. S. Gamini, "A conceptual framework on individual investors' learning behavior in the context of stock trading: An integrated perspective," *Cogent Econ. Finance*, vol. 6, no. 1, Jan. 2018, Art. no. 1544062.
[28] H. Yao, "The influence of price limits change on stock market based on simulation method," *Int. J. Social Sci. Econ. Res.*, vol. 5, no. 2, pp. 363–372, Feb. 2020.
[29] S. Thurner, J. D. Farmer, and J. Geanakoplos, "Leverage causes fat tails and clustered volatility," *Quant. Finance*, vol. 12, no. 5, pp. 695–707, May 2012.
[30] T. Mizuta, Y. Noritake, S. Hayakawa, and K. Izumi, "Affecting market efficiency by increasing speed of order matching systems on financial exchanges–investigation using agent based model," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–8.
[31] D. Byrd, M. Hybinette, and T. H. Balch, "ABIDES: Towards high-fidelity market simulation for AI research," 2019, *arXiv:1904.12066*. [Online]. Available: http://arxiv.org/abs/1904.12066
[32] M. Karpe, J. Fang, Z. Ma, and C. Wang, "Multi-agent reinforcement learning in a realistic limit order book market simulation," 2020, *arXiv:2006.05574*. [Online]. Available: http://arxiv.org/abs/2006.05574
[33] L. Ponta, S. Pastore, and S. Cincotti, "Static and dynamic factors in an information-based multi-asset artificial stock market," *Phys. A, Stat. Mech. Appl.*, vol. 492, pp. 814–823, Feb. 2018.
[34] M. Kvalvær and A. Bjerkøy, "Replicating financial markets using reinforcement learning; an agent based approach," Master's thesis, NTNU, 2019.
[35] I. Maeda, D. deGraw, M. Kitano, H. Matsushima, H. Sakaji, K. Izumi, and A. Kato, "Deep reinforcement learning in agent based financial market simulation," *J. Risk Financial Manage.*, vol. 13, no. 4, p. 71, Apr. 2020.
[36] J. Li, X. Wang, Y. Lin, A. Sinha, and M. Wellman, "Generating realistic stock market order streams," *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 01, pp. 727–734, Apr. 2020.
[37] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*. [Online]. Available: http://arxiv.org/abs/1411.1784
[38] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017, *arXiv:1701.07875*. [Online]. Available: http://arxiv.org/abs/1701.07875
[39] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *J. Econometrics*, vol. 31, no. 3, pp. 307–327, Apr. 1986.
[40] D. P. Helmbold, R. E. Schapire, Y. Singer, and M. K. Warmuth, "On-line portfolio selection using multiplicative updates," *Math. Finance*, vol. 8, no. 4, pp. 325–347, Oct. 1998.
[41] A. Agarwal, E. Hazan, S. Kale, and R. E. Schapire, "Algorithms for portfolio management based on the Newton method," in *Proc. 23rd Int. Conf. Mach. Learn. ICML*, 2006, pp. 9–16.
[42] A. Borodin, R. El-Yaniv, and V. Gogan, "Can we learn to beat the best stock," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, pp. 345–352.
[43] B. Li, P. Zhao, S. C. H. Hoi, and V. Gopalkrishnan, "PAMR: Passive aggressive mean reversion strategy for portfolio selection," *Mach. Learn.*, vol. 87, no. 2, pp. 221–258, May 2012.
[44] B. Li and S. C. H. Hoi, "On-line portfolio selection with moving average reversion," 2012, *arXiv:1206.4626*. [Online]. Available: http://arxiv.org/abs/1206.4626
[45] L. Gao and W. Zhang, "Weighted moving average passive aggressive algorithm for online portfolio selection," in *Proc. 5th Int. Conf. Intell. Hum.-Mach.Syst. Cybern.*, Aug. 2013, pp. 327–330.
[46] A. Borodin, R. El-Yaniv, and V. Gogan, "On the competitive theory and practice of portfolio selection," in *Proc. Latin Amer. Symp. Theor. Informat.* Berlin, Germany: Springer, 2000, pp. 173–196.

CHIA-HSUAN KUO received the B.S. degree in information management from Tamkang University, New Taipei City, Taiwan, in 2018, and the M.S. degree in information management from National Chiao Tung University, Hsinchu, Taiwan, in 2020. Her research interests include reinforcement learning and quantitative trading.

CHIAO-TING CHEN received the B.S. degree in information management and finance and the M.S. degree in information management from National Chiao Tung University, Hsinchu, Taiwan, in 2018 and 2020, respectively, where she is currently pursuing the Ph.D. degree in computer science. Her research interests include deep learning, artificial intelligence, knowledge graph, graph embedding, and financial technology.

SIN-JING LIN received the B.S. degree in finance from National Taiwan University, Taipei City, Taiwan, in 2019. She is currently pursuing the M.S. degree in information management with National Chiao Tung University, Hsinchu, Taiwan. Her research interests include deep learning, artificial intelligence, knowledge graph, graph embedding, and financial technology.


SZU-HAO HUANG (Member, IEEE) received the B.E. and Ph.D. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2001 and 2009, respectively. He is currently an Assistant Professor with the Department of Information Management and Finance and the Chief Director with Financial Technology (FinTech) Innovation Research Center, National Chiao Tung University, Hsinchu. He has authored more than 50 papers published in the related international journals and conferences. His research interests include artificial intelligence, deep learning, recommender systems, computer vision, and financial technology. He is also the Principal Investigator of the MOST Financial Technology Innovation Industrial-Academic Alliance and several cooperation projects with leading companies in Taiwan.

• • •