Cost-Efficient Reinforcement Learning for Optimal Trade Execution on Dynamic Market Environment

Di Chen*
Cornell University
Ithaca, NY, USA
di@cs.cornell.edu

Yada Zhu†
MIT-IBM Watson AI Lab, IBM Research
Cambridge, MA, USA
yzhu@us.ibm.com

Miao Liu
IBM Research
Yorktown Heights, NY, USA
miao.liu1@us.ibm.com

Jianbo Li
Three Bridges Capital
New York, NY, USA
jianboliru@gmail.com

ABSTRACT

Learning a high-performance trade execution model via reinforcement learning (RL) requires interaction with the real dynamic market. However, the massive interactions required by direct RL would result in a significant training overhead. In this paper, we propose a cost-efficient reinforcement learning (RL) approach called Deep Dyna-Double Q-learning (D3Q), which integrates deep reinforcement learning and planning to reduce the training overhead while improving the trading performance. Specifically, D3Q includes a learnable market environment model, which approximates the market impact using real market experience, to enhance policy learning via the learned environment. Meanwhile, we propose a novel state-balanced exploration scheme to solve the exploration bias caused by the non-increasing residual inventory during the trade execution to accelerate model learning. As demonstrated by our extensive experiments, the proposed D3Q framework significantly increases sample efficiency and outperforms state-of-the-art methods on average trading cost as well.

CCS CONCEPTS

• **Computing methodologies** → **Sequential decision making**.

KEYWORDS

cost-efficient, sample efficiency, optimal trade execution, dynamic market

ACM Reference Format:
Di Chen, Yada Zhu, Miao Liu, and Jianbo Li. 2022. Cost-Efficient Reinforcement Learning for Optimal Trade Execution on Dynamic Market Environment. In *3rd ACM International Conference on AI in Finance (ICAIF '22), November 2–4, 2022, New York, NY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3533271.3561761

*The work was done during the first author's internship at IBM Research.
†To whom correspondence should be addressed.

ICAIF '22, November 2–4, 2022, New York, NY, USA

ACM ISBN 978-1-4503-9376-8/22/11…$15.00
https://doi.org/10.1145/3533271.3561761

1 INTRODUCTION

In the modern equity market, financial institutions frequently need to dramatically change the size of their holding of a given stock to rebalance their portfolio or meet risk tolerance. However, doing so in one large instantaneous trade may often be too costly or even impossible due to the limited supply of market liquidity for an asset. Therefore, market participants resort to slicing large trading orders into smaller child orders and executing them one by one during an extended period of time. Although each child order has a smaller quantity, they still result in changes to supply/demand equilibrium and cause adverse price movement in the dynamic market. This effect is known as *market impact* [18]. The market impact often causes the actual execution price for large orders to be worse than the initially observed price on the market. This difference in prices is the main source of trading costs [15]. To minimize trading costs, it is essential for financial institutions to take into account the market impact for optimal trade execution.

Optimal trade execution is a prospective research direction for reinforcement learning (RL) technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. [12] are the pioneers of the empirical application of RL to optimal trade execution problems. The work uses Q-learning which is based on tabular representations and linear function approximations to explain the relationship between market state variables and optimal actions, which might have limited representation power for high dimensional financial data. Recently, [2, 5, 10, 13] have applied more advanced RL techniques, such as Deep Q-networks (DQN), double DQN, and proximal policy optimization (PPO) to financial trading problems and shown better performance than standard approaches. However, all these works are developed under the static market assumption, where the market environments are represented by historical limit order books that do not change after trade execution and essentially do not take into account the *market impact*. As stated in [15], every average-sized order would have a non-negligible impact and change the market immediately. Therefore, the performance of RL agents trained with static historical data could degrade dramatically when they are applied to dynamic markets (we also justified this statement in our experiments). Moreover, existing works are

based on direct RL approaches (also called model-free RL [17]), which directly learn a policy from samples collected from agent-environment interactions. Direct RL approaches are usually simple, however, they may be sample inefficient and may require a large amount of agent-market interactions to learn a good policy, which leads to a significant training overhead when they are applied to real markets.

To bridge the gap of investigating RL for trade execution under a dynamic market environment and increasing sample efficiency to reduce the training overhead when it is applied to real markets, we propose a novel cost-efficient RL framework called Deep Dyna-Double Q-learning (D3Q), which integrates deep reinforcement learning and planning to reduce the training cost while saturating model's performance. Specifically, **(i)** D3Q includes a learnable market environment model, which approximates the market impact using real market experience, to enhance policy learning via the learned environment. **(ii)** Meanwhile, we propose a novel state-balanced exploration scheme to solve the exploration bias caused by the non-increasing inventory during the trade execution. By learning from both the learned environment and the real market interactions, we show that D3Q significantly increases the sample efficiency and outperforms existing methods. Our main contributions are:

- We explicitly consider and address the market dynamics for optimal trade execution filling the blank of existing works, which only work on static historical data.
- We developed a high-fidelity queue-reactive-based [6] virtual market built on historical data, to address the unaffordable training overhead on a real market.
- We propose D3Q, a hybrid RL approach that combines both direct reinforcement learning and planning, via a learnable market environment model, to increase the sample efficiency.
- We propose a novel state-balanced exploration scheme for trade execution tasks, solving the sample bias caused by the simple random exploration in the classical ϵ-greedy method.
- We conduct extensive experiments using multiple stocks and demonstrate that D3Q significantly increases sample efficiency and reduces trading cost due to market impact, outperforming existing methods for optimal trade execution.

2 RELATED WORK

Although a plethora of algorithmic trading algorithms have been developed based on both model-free RL techniques [2, 5, 10, 13] and advancement of deep learning based order book modeling of electronic trading systems [8, 22], most works only use historical data as a static market environment due to the unaffordable overhead of training on the real market. In addition, few papers consider bridging the gap between planning and reinforcement learning to effectively reducing sample complexity and taking trading cost into account for training. A noticeable work [20] studied a model-based RL for learning a trading policy. However that work only considered market environment model learning and trading policy learning as two separate steps, whereas our method, D3Q seamlessly integrates market environment model learning and trading policy learning in a close loop under the Dyna framework. Moreover, [20] only used historical trade prints as their RL agent's exploration action, which might limit the agent capability to discover useful information in states never visited before, and there is no guarantee that such a strategy allows all actions have equal chance to be executed if an agent's solely task is to liquidate an asset. While in our D3Q framework, we proposed a novel state-balanced exploration scheme to solve the exploration biased caused by the non-increasing residual inventory during the trade execution (liquidation) to accelerate model learning.

3 PROBLEM FORMULATION

3.1 Dynamic Market Environment

In today's order-driven exchange markets, market order (MO) and limited order (LO) are two major types of orders. MOs refer to orders requested to buy/sell immediately at the current market price. MOs are subject to large trading slippage (price change) due to execution speed requirements. LOs offer to buy/sell (equivalently, bid/ask) given quantities (volume) of an asset at no more/less than certain price limits. Due to the restricted price, LOs can lead to delayed fulfillment or an increased chance of unfulfillment. A collection of LOs at different price levels awaiting to be executed by counterpart MOs is called a limit order book (LOB). LOB provides rich information about the market status. Figure 1 shows a snapshot of a LOB, where $p_k \in \mathbb{R}^+$ and $v_k \in \mathbb{Z}^+$ denotes price and volume of LOs at level k, $k \in \mathbb{Z}^-$ and $k \in \mathbb{Z}^+$ correspond to buy and sell, and p_M is the mid-price given by arithmetic mean of the best ask price and best bid price. Following [6], we introduce a queue-reactive-based virtual market to represent the real-world market environment. To be specific, let $\mathbf{v} = (v_{-K}, ..., v_{-1}, v_1, ..., v_K) \in \mathbb{N}^{2K}$ represents the length of the queue formed by the LOs at each price level $k \in [-K, K]$. The dynamics of the queue is modeled as a continuous-time Markov jumping process. The queue size is approximated by the smallest integer that is larger than or equal to the volume available at the queue divided by the stock's average order size (denoted as AOS_k) at the corresponding level k. The queue length increasing rate (decreasing rate) is given by the intensity of new LOs (total intensity of new MOs and cancellation of LOs). The intensity of each of the three event types for level k is a function of v_k and is estimated based on queue-reactive model using historical LOBs over a long time horizon.

On the other hand, mid-price p_M changes if a MO or cancellation of a LO results in queue size at the best ask or best bid depletes to zero, i.e., $v_1 = 0$ or $v_{-1} = 0$; and the price at each level k shifts if new LOs of opposite side, i.e. buy for v_1 or sell for v_{-1}, are inserted in the LOB. Putting everything together, the virtual market explicitly considers the sophisticated market impact induced by executing the large orders from financial institutions as well as the probabilistic events of new market orders, new limit orders, and cancellation of limit orders. In this way, the virtual market provides a more realistic dynamic market environment than that represented by static LOBs in previous works for us to benchmark our D3Q framework and compare it with other baseline methods.

3.2 Reinforcement Learning for Order Execution

The large order execution problem requires an agent to liquidate a large number of equity shares, e.g., I unites, by the end of time

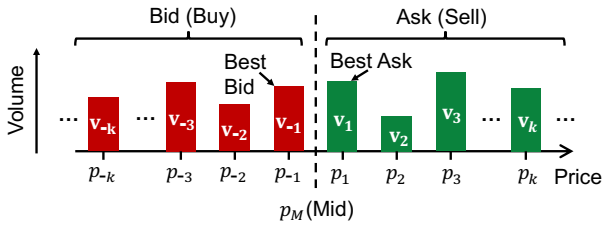

Figure 1: A snapshot of a LOB: the x-axis and y-axis corresponds to the limit order price and volume (quantity), respectively.

period $t = T$ while maximizing its trading profit. Let s_t, a_t denote the state and the action of the RL agent at time t, and $r(s_t, a_t)$ be the corresponding trading profit. An optimal trade execution policy maximizes the expected return from selling all the shares

$$\underset{a_0,a_1,...,a_T}{\text{argmax}} \ \mathbb{E}\left[\sum_{t=0}^{T} r(s_t, a_t)\right]$$

subject to trading cost due to market impact.

State space ($\mathcal{S}$): Each state $s \in \mathcal{S}$ is a vector of state attributes. The state space $\mathcal{S} = \mathcal{S}_{\text{agent}} \cup \mathcal{S}_{\text{market}}$ includes two subspaces: (i) the agent-state $s_a \in \mathcal{S}_{\text{agent}}$ consisting of *residual inventory* (rest_I), *residue time* (rest_T), and *decision price* p_D (the mid-price at $t = 0$); (ii) the market-state $s_m \in \mathcal{S}_{\text{market}}$ consisting of *queue size* $\mathbf{v}$ of LOB, *current mid-price* p_M, *signed volume* and *volatility* following [12].

Action space ($\mathcal{A}$): $\mathcal{A} = \{a | a \in [0, 1, 2, ..., \text{rest\_}I]\}$, i.e., selling a units of shares as a market order. We limit the action space to market orders to avoid the additional complication induced by the potential delay and the change of unfulfillment of limited orders.

Reward function: The reward function captures the immediate reward after an order has been executed, i.e.,

$$R(s, a) = (r(s, a) - p_D \cdot a) / p_D. \tag{1}$$

Intuitively, Eq.(1) evaluates the relative trading cost compared to selling a units of shares at the decision price p_D. p_D is given by p_M at the beginning of the trading period.

The goal in the RL framework for optimal trade execution is to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected return

$$G_t = \mathbb{E}_\pi[\sum_{k=0}^{T-t} \gamma^k R_{t+k} | s_t],$$

where $\gamma \in (0, 1]$ is a discount factor. Since trade execution problems are often performed in a short period of time, the discount factor is set to be 1. Herein we focus on a discrete setting for the trading time horizon. In the supplemental material, we discussed the potential and the poor performance of using an event-driven decision-making model with a continuous time horizon based on our exploratory study. In this paper, we learn the policy by estimating the action-value function $Q_\pi(s, a)$, given by

$$Q_\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{T-t} \gamma^k R_{t+k} | s_t = s, a_t = a]. \tag{2}$$

Notice that in large problems, it is infeasible to learn value for each state-action pair, so the Q-functions are often parameterized via a set of weights θ, i.e., $Q_\theta(s, a) \approx Q_\pi(s, a)$. Currently, neural networks are the most successful parameterization approach in RL such as DQN [11]. The DQN algorithm is a model-free RL approach that learns a policy purely based on real experiences. Although model-free approaches directly compute value functions, it might need a large amount of real-world experience to learn a good policy. Subsequently, it could lead to significant training overhead when the model is applied to real markets.

4 COST-EFFICIENT POLICY LEARNING VIA DEEP DYNA-DOUBLE Q (D3Q)

To better leverage real experience and increase sample efficiency, we propose a cost-efficient policy learning framework called Deep Dyna-Double Q-learning (D3Q). D3Q is inspired by the Dyna-Q framework [17], an instantiation of hybrid RL, which was developed based on tabular methods and was later extended to linear function approximations. Recently, Dyna-Q is also combined with deep learning approaches [14] leading to a framework called Deep Dyna-Q (D2Q) for building dialog systems. Motivated by the advances in RL technique, D3Q integrates the planning of the Dyna-Q scheme, the direct RL algorithm of double deep Q-learning, and a novel state-balanced exploration scheme to provide a cost-efficient hybrid algorithm for optimal trade execution policy learning.

Figure 2 depicts the overview of D3Q framework. Starting with an initial trade execution policy and an initial market environment model, the D3Q agent is trained in three processes: (1) direct RL, where the agent uses a state-balanced exploration scheme to interact with the real market, collects real experience, and improves trade execution policy using real experience; (2) environment model learning, where the market environment model is updated using the real experience explored to learn the market dynamics; and (3) planning, where the agent improves the trade execution policy using the simulated experience obtained from the learned market environment model. The implementation details are provided in the following subsections.

4.1 Direct RL with Deep Double Q-learning

We approximate the state-action value function (Q-function) via a deep Q-network (DQN) to estimate the expected cumulative return for an agent to perform an action in a given state. To avoid overestimating the Q-values, we adopt the double Q-learning method [4]. Specifically, we decouple the target Q-value of the next state into action selection and action evaluation by using the main Q-network Q_θ to select the best action, and the target network Q_ϕ to estimate the Q-value. Thus, the target Q-value for double Q-learning is

$$y_i = R^{(i)} + \gamma Q_\phi(s'^{(i)}, \underset{a'}{\text{argmax}}\, Q_\theta(s'^{(i)}, a')), \tag{3}$$

and the corresponding loss function becomes

$$L(\theta) = \frac{1}{B}\sum_{i=1}^{B}(y_i - Q_\theta(s^{(i)}, a^{(i)}))^2, \tag{4}$$

where $\{s^{(i)}, a^{(i)}, R^{(i)}, s'^{(i)}\}_{i=1}^{B}$ is a mini-batch sampled from a replay buffer $\mathcal{D}$, θ is the parameter of the main network, and ϕ is

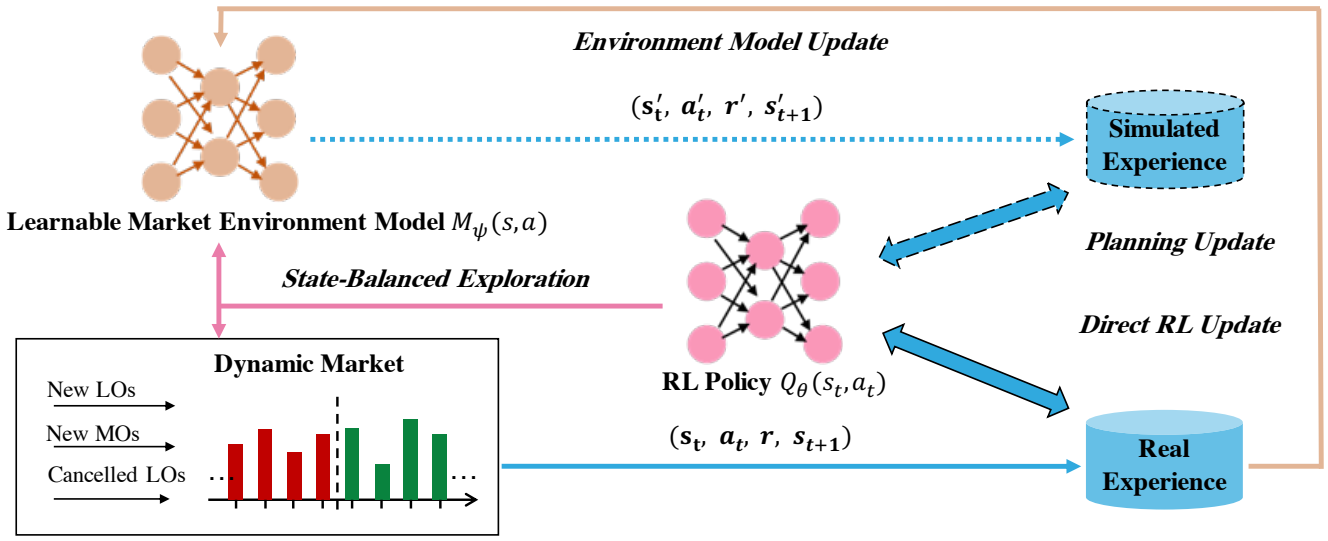

Figure 2: Overview of Deep Dyna-Double Q learning. Starting with an initial trade execution policy and an initial market environment model, the D3Q agent is trained in three processes: (1) direct RL, where the agent uses a state-balanced exploration scheme to interact with the real market, collects real experience, and improves trade execution policy using real experience; (2) environment model learning, where the market environment model is updated using the real experience explored to learn the market dynamics; and (3) planning, where the agent improves the trade execution policy using the simulated experience obtained from the learned market environment model. See algorithm 1 for more details.

the parameter of the target network. In our experiments, we use a "soft" target network update [9] to stabilize the training, by which rather than directly copying the weights after several iterations, the weights of the target network are updated by having them slowly track the learned networks, i.e., $\phi = \tau\theta + (1-\tau)\phi$ with $\tau = 0.01$ in our setting. We use a fixed-size first-in-first-out (FIFO) replay buffer to uniformly sample transitions, making samples less correlated.

4.2 State-Balanced Exploration for Trade Execution Problem

Sample bias caused by the simple uniform random exploration strategy for trade execution: Classical RL algorithms often use ϵ-greedy scheme to explore new experiences, where the agent starts exploration from an initial state and either take actions with the current policy or take a random action till the end of the episode. In the optimal trade execution task, the residual inventory (rest_I) is not only an important part of the agent-state but also confines the action space. Thus, unlike many other domains such as Atari games, where the action space remains invariant (except some corner cases) over time, the agent has a non-increasing action space in the optimal trade execution task, i.e., the agent can only choose to sell $a \in [0, 1, 2, ..., \text{rest\_}I]$ (rest_I is non-increasing) units of shares to make a valid action. Moreover, the agent also has to sell all residual shares by the end of the time period ($t = T$) to fulfill the trade execution task. Therefore, a simple random exploration scheme would cause a huge bias in the action distribution of generated experiences. Specifically, let us use a_t ($t = 0...T$) to denote the action (the number of units of shares sold) at time t. If we apply a random exploration scheme that uniformly chooses an action between 0 and rest_I_t (the residual inventory at time t, rest_$I_0 = I$), then we have

$$\mathbb{E}[a_t] = I/2^{t+1} \text{ and } \mathbb{E}[\text{rest\_}I_t] = I/2^t. \quad (5)$$

Intuitively, a uniform random exploration would on average sell half of the residual inventory at each step and therefore would sell out all shares in the first several steps ignoring the potential benefits of selling more shares later. Therefore, the uniform random exploration steps in ϵ-greedy scheme make it hardly explore the experience that the agent liquidates most inventory in the later time steps.

State-balanced exploration scheme: To solve the sample bias caused by the simple random exploration scheme, we propose the *state-balanced sampling process*, which provides a uniform sampling for all possible valid trade execution plan $(a_0, ..., a_T)$ such that $\sum_{t=0}^{T} a_t = I$.

Proposition 4.1. *Given the residual inventory I and residual time steps T, we sample T distinct integers from a uniform distribution over integers 1 to $I+T$ and sort them into an ascending sequence z_i ($i \in [1..T]$ and $z_i < z_{i+1}$). Let $z_0 = 0$ and $z_{T+1} = I+T+1$, we can compute $a_t = z_{t+1} - z_t - 1$ ($t \in [0..T]$). Then, the sequence $(a_0, ..., a_T)$ is a valid trade execution plan with $\sum_{t=0}^{T} a_t = I$ and $a_t \geq 0$. Moreover, for any two valid trade execution plans $(a_0, ..., a_T)$ and $(a'_0, ..., a'_T)$, they have an equal chance of being sampled via this sampling process.*

Proof. Given the equation $\sum_{t=0}^{T} a_t = I$, we have $\sum_{t=0}^{T}(a_t + 1) = I + T + 1$. By *stars and bars* [21], one valid sequence $(a_0, ..., a_T)$ corresponds to a partition plan of $I+T+1$ stars (which has $I+T$ gaps in between) using T bars. The configurations of bars are one-to-one and onto mapped with the valid sequences. Therefore, we

can uniformly sample the valid sequences by uniformly sample the locations of bars. Thus, our sampling process provides a uniform sampling for all valid trade execution plan. □

Subsequently, we can replace the simple uniform random action sampling steps in the classical ϵ-greedy by our *state-balanced sampling process* depicted in Proposition 4.1 to form the state-balanced exploration scheme. Note that, not only can the *state-balanced sampling process* be used to sample all random actions at the beginning of exploration, it can also be used to sample a one-step random action given the residual inventory and the residual time steps (by taking the first action only). As shown in our experiments, not only state-balanced exploration scheme better explores the action space but also improves the performance of the learned policy compared with the policy learned using the ϵ-greedy scheme (D3Q(0) vs. DDQN).

4.3 Planning with the Learnable Environment Model

In order to improve sample-efficiency and reduce training cost, we developed a learnable market environment model to mimic market dynamics and generate the simulated experience that can be used to improve the trade execution policy.

Specifically, we use a neural network $M_\psi(s, a)$ parameterized by ψ to predict the state s' at the next time step after executing action a. Since the transition of the agent state is fixed given the chosen action (the *residual time* is always reduced by one, the *residual inventory* decreases by a units of shares, and the decision price p_D keeps the same from the beginning of the episode), predicting the next state is essentially a regression task of simulating the next market state s'_m(i.e., market statistics and the volumes of different levels of LOB). Note that, we do not need to predict the reward since it can be easily computed using the current market state and the chosen action.

The environment model $M_\psi(s, a)$ is learned using real experience (s_t, a_t, r, s_{t+1}) sampled from the replay buffer $\mathcal{D}$ and is optimized with L2-loss. We used the same learning rate and batch size as the policy model to train the environment model. Moreover, to avoid the potential error accumulation of the learned environment, we do not generate a sequence of experience of a full trading episode from a sampled initial state. Instead, we only generate one-step simulated experience starting from a real state s, and use the state-balanced exploration scheme to sample a one-step action a for the starting real state s. Then we compute the corresponding reward r based on the market state in s and use the environment model to generate the next state s'.

In D3Q, we interleave the updates of our policy using the real experience and the updates using the simulated experience obtained from the learned market environment model. As shown in our experiments, by performing multiple updates using the simulated experience per update using the real experience, we can significantly speed up our policy learning without degrading the final performance of our learned policy, which reduces the training overhead when it is applied to the real market. See algorithm 1 for the pseudo-code of Deep Dyna-Double Q learning.

Algorithm 1: Deep Dyna-Double Q (D3Q) Optimal Trade Execution Method

Input: learning rates α^θ, exploration probability ϵ, target network soft update ratio τ, maximum inventory level N, planning update frequency K, trading period T
Output: θ,ψ
Initialize real experience replay buffer $\mathcal{D}$ using state-balanced random exploration scheme on the dynamic market.
Initialize the Q-network $Q_\theta(s, a)$ and the environment model $M_\psi(s, a)$ via pre-training on the real experience from the replay buffer $\mathcal{D}$. Initialize $Q_\phi(s, a)$ with $\phi = \theta$.
for i=1, ..., N_{episodes} **do**
    Explore new experience using state-balanced exploration scheme on the dynamic market.
    for j=1, ..., K **do**
        Update $Q_\theta(s, a)$ using experience generated by the environment model $M_\psi(s, a)$.
    end
    Update $Q_\theta(s, a)$ using real experience sampled from the replay buffer $\mathcal{D}$.
    Update the environment model $M_\psi(s, a)$ using real experience sampled from the replay buffer $\mathcal{D}$.
    Update the parameters of the target network with $\psi = (1 - \tau) \cdot \psi + \tau \cdot \theta$
end

5 EXPERIMENT

In this section, we evaluate the proposed D3Q RL framework with respect to market impact cost, exploration cost, and compare its performance with state-of-the-art approaches for trade execution.

5.1 Setup

We select seven stocks including AAPL, HAL, HPQ, HSBC, PFE, SBUX, and VZ from six different market sectors. All of them are high-liquid stocks that the average market volume ranges from two to one hundred fifteen million shares per day. We obtain their limited order book data from March 2020 to December 2020 from the Investors Exchange stock market (IEX). As described in section 2.1, we use a queue-reactive-based virtual market to represent the real dynamic market environment. To be specific, we set the time step to be one second and use the average order size of level-1 (AOS_1) as the trade execution unit. Given this setting, we calculate the intensity of new MOs, new LOS, and cancellation of LOs (See [6] for more details) that determines the queue length increasing rate (decreasing rate) and the shift of price at each level k. Considering the liquidity of the selected stocks, we set the initial inventory to be 80 trade execution units ($I = 80$) and the number of time steps to sell all shares to be 16 ($T = 16$). We use the average trading cost (ATC) as the performance metrics, which measures the average relative trading cost compared with selling all stocks at the decision-price p_D, i.e.,

$$\text{ATC} = (\text{The average selling price} - p_D)/p_D.$$

Thus, ATC is essentially a regret value w.r.t. the decision price at the beginning of the trading horizon. Note that, we use the basis point (0.01%) as the unit of ATC and ATC is often a negative number due to market impact (the price drops down when you

Models \ Stocks	AAPL	HAL	HPQ	HSBC	PFE	SBUX	VZ
D3Q(0)	-12.89±0.13	-18.83±0.15	-18.14±0.12	-14.40±0.12	-14.85±0.14	-13.17±0.07	-16.92±0.16
D3Q(7)	**-11.53±0.14**	**-18.23±0.14**	**-17.98±0.11**	**-14.28±0.10**	**-14.55±0.15**	**-12.81±0.09**	**-16.77±0.19**
SAC	-15.79±0.12	-21.69±0.12	-18.74±0.11	-15.53±0.11	-16.37±0.17	-13.24±0.08	-18.56±0.16
DDQN	-13.40±0.14	-19.24±0.13	-18.21±0.11	-14.85±0.11	-16.63±0.13	-13.22±0.08	-19.10±0.16
QL-Linear	-16.73±0.13	-21.52±0.14	-21.06±0.13	-17.00±0.10	-18.95±0.15	-14.24±0.08	-19.73±0.16
QL-Tabular	-17.23±0.16	-22.25±0.12	-21.47±0.11	-18.07±0.10	-19.30±0.13	-14.83±0.03	-21.46±0.14
First All	-26.55±0.11	-32.06±0.12	-25.50±0.11	-21.39±0.07	-28.60±0.12	-15.06±0.03	-32.86±0.13
Last All	-27.81±0.14	-33.50±0.15	-26.42±0.13	-22.59±0.10	-30.99±0.28	-15.23±0.03	-35.57±0.23
Equal Split	-16.69±0.12	-22.66±0.12	-22.05±0.11	-18.33±0.09	-19.21±0.12	-15.32±0.04	-22.18±0.14
VWAP	-16.72±0.14	-22.53±0.12	-21.77±0.12	-18.21±0.09	-19.32±0.12	-15.29±0.05	-21.97±0.15

Table 1: Performance (ATC 0.01%, the smaller the absolute value the better) comparison of different models. D3Q significantly outperforms both other RL methods and classical trading policies.

are liquidating your stocks). To cancel the variation caused by the market, the performance of all models is calculated using 1,000 trading tests with different random seeds to report the mean and the standard deviation. For benchmark models, we follow their original configuration and exhaustively explore the hyper-parameters to saturate their performance.

5.2 Implementation Details

In our experiments, we implemented the Q-network in D3Q using a dueling architecture [19], where we used two 3-layer-fully-connected networks with the number of hidden units 512, 256, 128, 1 and 512, 256, 128, N_inventory (the total inventory to liquidate), respectively, to model the state value function $V(s)$ and the advantage function $A(s, a)$. For the market state, we considered 20 market variables in our experiments including the volumes of 16 levels of the current LOB, the current reference price p_R, the initial decision price p_D, the market volatility, and the total volume in the LOB. Accordingly, we used a 3-layer-fully-connected network with the number of hidden units 512, 256, 128, 20 as the environment model to learn to regress the market state using the L2 loss. All experiments were performed on one NVIDIA Tesla K40m GPU with 4GB memory. For all models in our experiments, the training process was done for 200,000 episodes with a random exploration rate $\epsilon = 0.3$, using a batch size of 64, and an Adam optimizer [7] with a learning rate of 0.0001. The training process took about 20 hours to finish and the testing process was done for 1000 runs of the trading test, which took about 2 minutes for each.

5.3 Result and Analysis

5.3.1 Market impact plays an important role. Most existing works for optimal trade execution use historical data as their training set and assume that their own execution is negligible to the whole market. However, as stated in [15], every average-sized order would have a non-negligible impact and change the market immediately. Therefore, to investigate whether the market impact is negligible or not, we compared the test performance (in our dynamic market environment) of the D3Q trained with the dynamic market environment and the D3Q trained without market impact (named

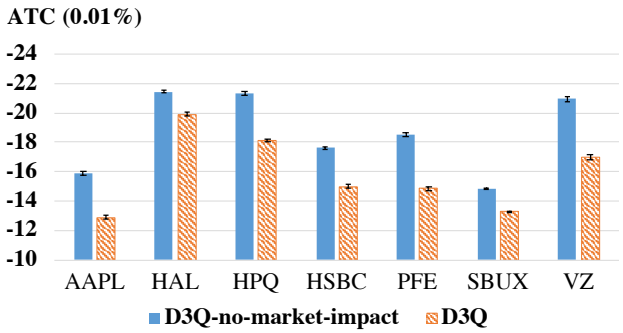

Figure 3: Performance Comparison between the D3Q trained with static historical data and the D3Q trained with dynamic market environment (the lower the better).

D3Q-no-market-impact). Specifically, for the training of the D3Q-no-market-impact, we let the virtual market generate the next step limit order book without taking into account the action taken by the agent. Therefore, the model is trained using "historical data" generated by our virtual market ignoring the market impact of its actions. To focus on the market impact comparison, we turned off the planning module (ignoring the sample efficiency) in D3Q framework and only train them with real experience. As shown in Figure 3, the performance of D3Q for all the seven stocks degrades significantly when the market impact is ignored, on average losing 3.04 extra basis points. This result justifies the importance of considering market impact for training a high-performance trading agent for the optimal trade execution problem.

5.3.2 Performance comparison with benchmark methods. To demonstrate the advantage of our state-balanced exploration scheme and benchmark the performance of D3Q, we compared D3Q with eight optimal trade execution baselines: 1) Soft Actor-Critic (**SAC**) [3]: an off-policy actor-critic deep RL algorithm based on the maximum entropy reinforcement; 2) Double Deep Q-learning models (**DDQN**) [4]: a variant of deep Q-learning algorithm that alleviates the over-estimation problem; 3) Tabular Q-learning methods (**QL-Tabular**) [12]; 4) Q-learning using linear tile coding (**QL-Linear**) [16]; 5)

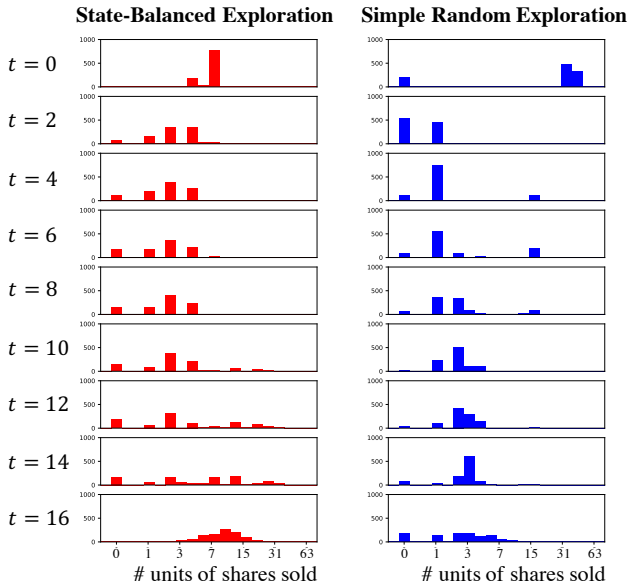

Figure 4: The probability distribution of actions (log-scale) taken at different time steps. The histogram is collected from 1,000 trading tests on the virtual market of AAPL using the agents trained with D3Q (state-balanced exploration scheme) and DDQN (simple ϵ-greedy random exploration) respectively. Limited by space, we only show the histogram every other step.

First All: selling all stocks at the beginning; 6) **Last All**: selling all stocks at the end; 7) **Equal Split**: time-weighted average price during the order execution; 8) Market volume-weighted average price (**VWAP**): "Equal Split" orders during the period of order executing.

Soft Actor-Critic (SAC) and Double Deep Q-learning (DDQN) are the most advanced RL approaches in literature for optimal trade execution. QL-Tabular and QL-Linear are standard RL-based methods for this type of problem, which uses tabular function and linear model respectively for learning the Q-function. The rest are classical trade execution strategies that are commonly used in practice. In particular, TWAP can be proved to be the optimal strategy when agents believe that the asset price process is a martingale [1]. Here, we saturated all models' performance by training all of them with massive real experience (200,000 real exploration episodes) in our dynamic virtual market. For the proposed D3Q approach, we evaluate two settings, i.e., the basic setting without planning (named D3Q(0)) and the setting with 7 planning updates per real experience update (named D3Q(7)). The results are summarized in Table 1, where we have the following observations. D3Q achieves a better average ATC trading cost than all the model-free RL baselines as well as classical trade execution strategies across the seven stocks. In addition, D3Q significantly outperforms TWAP, suggesting that D3Q can capture more complicated market dynamics. Moreover, with the same amount of real experience, D3Q's performance is further boosted using the simulated experience (D3Q(7)), which explores more possible execution plans.

On the other hand, by comparing the performance between D3Q and DDQN, we demonstrate the advantage of our state-balanced exploration scheme over the simple ϵ-greedy exploration strategy. As an illustrative example, we visualize the policies learned by D3Q and DDQN on the stock of Apple Inc. (AAPL). As shown in Figure 4, due to the exploration bias of the simple random exploration strategy, the agent trained by DDQN tends to sell about half of the inventory ($I = 80$) at the first several steps leaving few actions for the following steps. In contrast, the agent trained with D3Q has a more 'patient' trading strategy that can even liquidate a significant portion of shares in the last several steps, which leads to better overall performance compared with DDQN.

5.3.3 Reducing exploration cost via planning. To illustrate how the planning module can reduce the training cost, we compare the convergence behavior of D3Qs trained with different planning update frequency. We keep the same setting for the total time steps and the total trading volume ($I = 80$, $T = 16$), and vary the number of planning updates per real experience update. We use the name D3Q(n) to denote the D3Q model trained with n planning updates per real experience update (D3Q(0) denotes the model trained without planning update). We present the test performance trajectories of two selected stocks (AAPL and VZ) in Figure 5. Note we observe similar patterns for other stocks and skip the plots of other stocks due to space limitations. As shown in Figure 5, D3Q(7) reaches -13 (-17) ATC within 30,000 (20,000) real market interactions while it takes about 100,000 (70,000) real market interactions for D3Q(0) to achieve the same ATC value. Therefore, the planning module can significantly increase the convergence rate leading to a lower training cost when it is applied to real markets. Putting together the observations from Table 1 and Figure 5, we can see that the proposed D3Q RL framework for optimal trade execution not only reduces the market impact cost but also increases sample efficiency in training.

5.4 Discussion of the Event-Driven Decision-Making Model

In this work, we adopted a discrete setting for the trading time horizon, where the actions are executed at the end of each time step (second). However, in reality, market status can change rapidly between time steps for highly liquid stocks. Ideally, people want to have an event-driven decision-making model that can catch such a rapidly changing market and respond to each market change at any time within the trading horizon. Specifically, during the time period $t \in [0, T]$, a trade action with quantity $a \in$[0, 1,..., number of residual inventory] is always executed whenever LOB changes. Consequently, the next state $s'^{(i)}$ in equation (3) changes from the state at the end of the next time step to the state when the next market change happens.

Motivated by the potential advantage of the event-driven decision-making model described above, we explored and experimented an event-driven variant of our D3Q model. However, we observe that the ATC of the *event-driven D3Q model* is consistently worse than that of the discrete-time D3Q. This surprising result is mainly because: 1) the number of decisions increases dramatically within the given time horizon and varies from time to time due to the stochastic market behavior; 2) the effective actions ($a > 0$) are

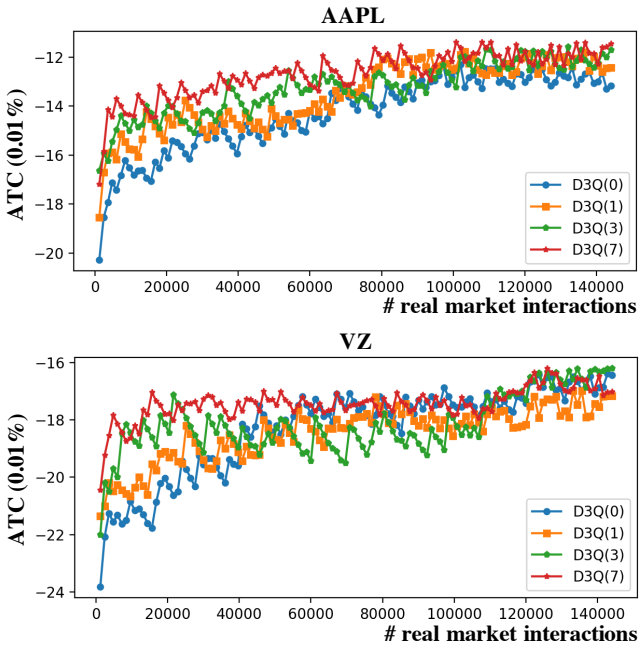

Figure 5: Convergence behavior of D3Q on the stock AAPL and VZ, trained with different planning steps (the quicker the better).

very sparse where the model is expected to have no action most of the time. Thus, despite the event-driven decision-making model allows finer granularity decisions, the significantly increased decision space and complexity results in a slower convergence and an inferior performance. Thereupon, here we focus on the discrete-time decision-making model.

6 CONCLUSION

In this paper, we propose a hybrid RL framework D3Q that integrates deep reinforcement learning and planning to optimal trade execution under dynamic market environments. To address the blank of optimal trade execution research on dynamic markets, we developed a high-fidelity queue-reactive based virtual market to address the unaffordable training overhead of using a real market. Moreover, we demonstrate that the D3Q framework significantly increases sample efficiency, reduces the market impact cost, and outperforms existing methods. In the future, we will extend this work to include more complicated situations in the real markets and eventually deploy our learned policies on the real market trading.

ACKNOWLEDGMENTS

This work was supported by the MIT-IBM Watson AI Lab and Wells Fargo Company through the Membership Program of the MIT-IBM Watson AI Lab. The views and conclusions expressed herein are those of the authors and should not be interpreted as that of IBM Research or Wells Fargo Company.

REFERENCES

[1] R. Almgren and N. Chriss. 2001. Optimal execution of portfolio transactions. *Journal of Risk* 3 (2001), 5–40.
[2] Kevin Dabérius, Elvin Granat, and Patrik Karlsson. 2019. Deep Execution - Value and Policy Based Reinforcement Learning for Trading and Beating Market Benchmarks. *SSRN Electronic Journal* (01 2019). https://doi.org/10.2139/ssrn.3374766
[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* (2018).
[4] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, 2094–2100.
[5] Chien Yi Huang. 2018. Financial Trading as a Game: A Deep Reinforcement Learning Approach. *arXiv e-prints*, Article arXiv:1807.02787 (Jul 2018), arXiv:1807.02787 pages. arXiv:1807.02787 [q-fin.TR]
[6] Weibing Huang, Charles-Albert Lehalle, and Mathieu Rosenbaum. 2015. Simulating and analyzing order book data: The queue-reactive model. *J. Amer. Statist. Assoc.* 110, 509 (2015), 107–122.
[7] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[8] Junyi Li, Xintong Wang, Yaoyang Lin, Arunesh Sinha, and Michael Wellman. 2020. Generating Realistic Stock Market Order Streams. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 01 (Apr. 2020), 727–734. https://doi.org/10.1609/aaai.v34i01.5415
[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
[10] Siyu Lin and Peter A. Beling. 2020. An End-to-End Optimal Trade Execution Framework based on Proximal Policy Optimization. In *Proceedings of the 29th International conference on Artificial Intelligence.*
[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533. http://dx.doi.org/10.1038/nature14236
[12] Yuriy Nevmyvaka, Yi Feng, and Michael J. Kearns. 2006. Reinforcement learning for optimized trade execution. In *ICML*. 673–680. https://doi.org/10.1145/1143844.1143929
[13] Brian Ning, Franco Ho Ting Ling, and Sebastian Jaimungal. 2018. Double Deep Q-Learning for Optimal Execution. arXiv:1812.06600 [q-fin.TR]
[14] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. 2018. Deep Dyna-Q: Integrating Planning for Task-Completion Dialogue Policy Learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 2182–2192. https://doi.org/10.18653/v1/P18-1203
[15] Emilio Said, Ahmed Bel Hadj Ayed, Alexandre Husson, and Frédéric Abergel. 2017. Market Impact: A systematic study of limit orders. *Market Microstructure and Liquidity* 3, 03n04 (2017), 1850008.
[16] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. Market Making via Reinforcement Learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems* (Stockholm, Sweden) *(AAMAS '18)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 434–442.
[17] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA.
[18] N. G. Torre. 1997. *Market Impact Model Handbook.* BARRA Inc., Berkeley.
[19] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. 1995–2003.
[20] Haoran Wei, Yuanbo Wang, Lidia Mangu, and Keith Decker. 2019. Model-based reinforcement learning for predictions and control for limit order books. *arXiv preprint arXiv:1910.03743* (2019).
[21] Wikipedia. 2021. Stars and bars (combinatorics) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Stars%20and%20bars%20(combinatorics)&oldid=997315407. [Online; accessed 09-January-2021].
[22] Zihao Zhang and Stefan Zohren. 2021. Multi-Horizon Forecasting for Limit Order Books: Novel Deep Learning Approaches and Hardware Acceleration using Intelligent Processing Units. arXiv:2105.10430 [cs.LG]