Learn Continuously, Act Discretely: Hybrid Action-Space Reinforcement Learning For Optimal Execution

Feiyang Pan[*1], **Tongzhe Zhang**[*2], **Ling Luo**[1], **Jia He**[1] and **Shuoling Liu**[2]
[1]Huawei EI Innovation Lab
[2]E Fund Innovation Lab
pfy824@gmail.com, ztz1992@foxmail.com

Abstract

Optimal execution is a sequential decision-making problem for cost-saving in algorithmic trading. Studies have found that reinforcement learning (RL) can help decide the order-splitting sizes. However, a problem remains unsolved: how to place limit orders at appropriate limit prices?

The key challenge lies in the "continuous-discrete duality" of the action space. On the one hand, the continuous action space using percentage changes in prices is preferred for generalization. On the other hand, the trader eventually needs to choose limit prices discretely due to the existence of the tick size, which requires specialization for every single stock with different characteristics (e.g., the liquidity and the price range). So we need continuous control for generalization and discrete control for specialization. To this end, we propose a hybrid RL method to combine the advantages of both of them. We first use a continuous control agent to scope an action subset, then deploy a fine-grained agent to choose a specific limit price. Extensive experiments show that our method has higher sample efficiency and better training stability than existing RL algorithms and significantly outperforms previous learning-based methods for order execution.

1 Introduction

Since computer algorithms encompass the whole trading process in modern financial markets, optimal execution has attracted significant research attention for decades. For the buy-side, optimal execution aims to save cost and reduce market impact when executing large orders. It is often viewed as a stochastic sequential decision-making process because large orders need to be divided into a series of small sub-orders within a period of time to be filled without influencing the market too much. Traders or algorithmic trading systems should decide the prices and volumes of the split sub-orders during this process.

There are two fundamental types of transactions: market orders and limit orders. Market orders are transactions meant to be executed as quickly as possible at the market price. Therefore, when using market orders, the agent only decides the size of each sub-order. Many studies showed that modern RL is powerful for such a problem [Nevmyvaka *et al.*, 2006; Hendricks and Wilcox, 2014; Ning *et al.*, 2018; Fang *et al.*, 2021].

However, there are two disadvantages for such methods: 1) market orders have no control over the price and might be filled at disappointing prices if the bid-ask spreads are wide, and 2) the volume of sub-orders given by RL agents might fluctuate too much because they work as a sort of market timing, potentially causing large market impact. So in this paper, in contrast to previous work, we focus on optimal execution with limit orders rather than market orders.

A limit order is an order to buy or sell a stock at a specific price or better, which can only be filled if the stock's market price reaches the limit price. Therefore, an agent for optimal limit order placement should control both limit prices and volumes of sub-orders.

A challenge emerges when using a learning-based agent to decide the limit price: the action space can be viewed as both discrete and continuous. From the perspective of the original problem, the choices of limit prices are discrete due to the existence of the tick size (the minimum price increment). For example, for a \$10.00 stock, one may set bid prices at \$10.00 or \$9.99, but not at \$9.995. So an ideal end-to-end agent should use discrete control to select the actual prices precisely, but it is usually impossible to generalize between stocks with different price-level and liquidity. For example, an action of -1 tick means -1% for a \$1.00 stock but -0.01% for a \$100.00 stock.

On the other hand, people usually use percentages to understand the rise and fall of stock prices, which is a continuous representation. As training deep RL agents with high-dimensional and noisy state dynamics requires a lot of trial-and-error, continuous control that determines the percentage change is preferred for its generalization ability across stocks with different prices. For example, a bid at -0.1% results in \$9.99 for a \$10.00 stock, and results in \$99.90 for a \$100.00 stock. However, such a method is more inclined to learn the strategy of stocks with high prices and liquidity, and fails to specialize in determining the ticks for illiquid low-priced stocks.

In light of these observations, we propose Hybrid Action-space Limit Order Placement (HALOP), which trains an

agent to learn continuously and act discretely. First, for generalization, HALOP employs a continuous-control agent to scope an action subset with high-dimensional market dynamics as input. Next, a fine-grained discrete-control agent chooses a specific discrete limit price from the action subset, which specializes well in stocks with different characteristics. Both agents are trained end-to-end to maximize the long-term return, i.e., the excess return in execution.

We conduct extensive experiments and show that our new optimal execution method with limit orders can beat the market, and significantly improves the excess return upon previous learning-based order execution methods.

2 Related Work

2.1 Optimal Execution

Non-machine-learning Order Execution

Traditional optimal execution research tends to assume that the market price movements follow some stochastic process such as the Brownian motion, and then use stochastic optimal control method to derive the volume trajectory analytically [Almgren and Chriss, 2001; Huberman and Stanzl, 2000; Bertsimas and Lo, 1998]. However, practitioners rarely use these methods to place orders in stock markets, because the assumptions may not hold in the real-world. The most widely used trade execution strategies are based on pure rules or statistical rules. For example, the time-weighted average price (TWAP) [Bertsimas and Lo, 1998] strategy divides a large order into equal-sized sub-orders and executes each within equally divided time intervals.

The volume-weighted average price (VWAP) [Kakade *et al.*, 2004] strategy first estimates the average volume traded for each time interval from historical data, then splits order based on the estimates.

Although simple enough, TWAP and VWAP are still popular these days because their execution costs are always close to the market. Specifically, we let the TWAP strategy with market order be the benchmark in this paper.

Reinforcement Learning for Order Execution

As order execution is fundamentally a problem of making decisions under uncertainty and the actual trading data is full of noise, RL becomes the best choice to solve such a problem. Nevmyvaka [Nevmyvaka *et al.*, 2006] is the pioneer for RL in optimal execution where the agent is trained by Q-Learning [Watkins and Dayan, 1992] to chooses the limit price. However, they only considers a small limited set of discrete actions (a few bid and ask prices) without considering the more important percent changes in actions.

With the development of deep learning and deep RL in the past few years, some studies use deep RL to learn to execute orders with high-dimensional market data as inputs [Hendricks and Wilcox, 2014; Ning *et al.*, 2018; Lin and Beling, 2019; Lin and Beling, 2020; Fang *et al.*, 2021]. While [Hendricks and Wilcox, 2014] applies RL to modify a given volume trajectory suggested by the traditional order execution model Almgren-Chriss [Hendricks and Wilcox, 2014]. Without any market assumptions, several variations of Deep Q-Network (DQN) [Mnih *et al.*, 2013] are proposed to choose discrete volumes, which could address the high dimensions and the complexity of the finance market and trading signals with the deep neural network [Ning *et al.*, 2018; Lin and Beling, 2019]. Instead of manually designed attributes, a PPO-based optimal execution framework is designed to make decisions based on raw level-2 market data [Lin and Beling, 2020]. Policy distillation paradigm is also deployed in order execution [Fang *et al.*, 2021], which uses a distilled PPO agent to choose optimal order-splitting volumes. However, most of the previous work studies the problem of volume-splitting, which only addresses the order execution problem partially.

3 Hybrid Action-Space Optimal Execution

3.1 Problem Setting

In this part, we first describe the general setting of the optimal execution problem with limit orders, then formulate it as a sequential Markov Decision Process (MDP). In particular, we focus on setting the limit prices of sub-orders rather than the volumes for clarity, although the proposed method can work seamlessly together with previous volume-oriented methods.

General Problem Description

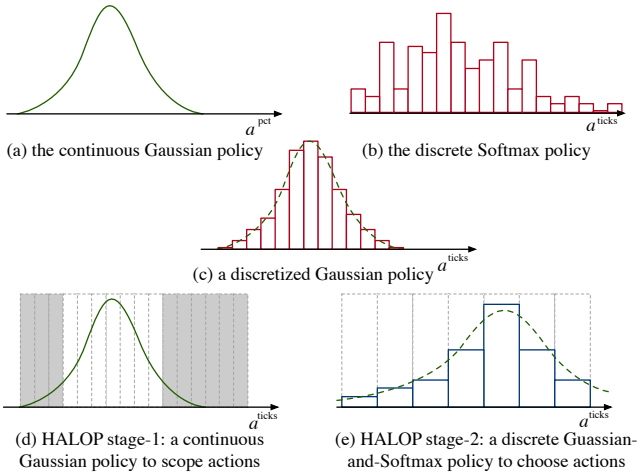
We simplify the problem as a discrete-time decision process following [Cartea *et al.*, 2015; Ning *et al.*, 2018; Fang *et al.*, 2021], where the agent interacts with the environment multiple times within a predefined time period based on discrete time space. So consider a horizon of T timesteps $t = 1, \dots, T$, each represents a $1/T$ of a predefined total time periods. For example, if there is a total time period of three hours and $T = 90$, it has 90 small time periods of two minutes. At the t-th step (the beginning of the t-th small time period), the environment reveals the recent market dynamics to the agent, consisting of a series of history quotes and top 5 bid/ask prices and volumes. Then, the agent sends a limit order with a limit price rounded to the tick size and a volume. Next, at the end of the $1/T$ period, the environment tells the agent whether (or how much) the order has been filled, and withdraw the rest of it. Then it moves to the $(t+1)$-th timestep. Specifically, at the end of the T-th period, if there is still inventory remained, the system automatically send a market order to execute it at once.

In addition, as we focus on the price, we assume there is a predefined *schedule* of trading volume for the agent, denoted by $v_1^*, \dots, v_T^*$, which sums to one $\sum_{t=1}^{T} v_t^* = 1$. For example, it can be the volumes of classic financial strategies like TWAP or VWAP, or the outputs of any volume-oriented algorithmic trading agent [Fang *et al.*, 2021]. As the limit orders are not guaranteed to be executed, let $\tilde{v}_1, \dots, \tilde{v}_T$ be the *executed* volume within each period, which has $\Delta_t = \sum_{\tau=1}^{t} v_\tau^* - \sum_{\tau=1}^{t} \tilde{v}_\tau \geq 0$. So at step t, we assume that the agent tries to catch up with the schedule, by placing a new order with a volume

$$v_t = v_t^* + \Delta_{t-1}. \tag{1}$$

MDP Formulations for Limit Order Placement

Now we formulate the problem as a finite-time MDP, and put forward the continuous-discrete duality of the problem.



(a) the continuous Gaussian policy (b) the discrete Softmax policy

(c) a discretized Gaussian policy

(d) HALOP stage-1: a continuous Gaussian policy to scope actions

(e) HALOP stage-2: a discrete Guassian-and-Softmax policy to choose actions

Figure 1: Illustration of different types of policies for different types of action-spaces.

The state consists of two types of information, the public state and the private state $s_t = (s_t^{\text{pub}}, s_t^{\text{priv}})$. Similar to [Nevmyvaka *et al.*, 2006; Lin and Beling, 2020; Fang *et al.*, 2021], the public state includes past market dynamics obtained from the environment. Private observations are real-time status of the running agent, including the remaining inventory $1 - \sum_{\tau=1}^{t-1} \tilde{v}_\tau$; the amount of unfulfilled orders Δ_{t-1}; and the remaining time fraction $1 - t/T$.

The action, by default, should be an actual price rounded to the tick size. For example, for a \$10.00 stock, the action of a bid should be like \$9.97. However, such an action changing over time cannot directly be used to formulate an MDP. So we put forward two types of action representations:

- Action measured by ticks $a_t^{\text{ticks}} \in \mathcal{A}^{\text{ticks}} \subset \mathbb{Z}$, i.e.,

  $$a_t^{\text{ticks}} = (\text{LimitPrice}_t - \text{CurrentPrice}_t)/\text{TickSize}$$

  It is natural to represent price changes in ticks based on the current price[1]. So it is a discrete number, e.g., a bid at \$9.97 for a \$10.00 can be represented as $a^{\text{ticks}} = -3$, if the tick size is one cent.

- Action measured by percentage $a_t^{\text{pct}} \in \mathcal{A}^{\text{pct}} \subset \mathbb{R}$, i.e.,

  $$a_t^{\text{pct}} = (\text{LimitPrice}_t/\text{CurrentPrice}_t - 1) \times 100\%$$

  It is more general and easier to understand when measuring price movements by percentages or basis points. So any action can has a continuous representation by basis points, which is a continuous variable. E.g., the same bid can be represented as $a^{\text{pct}} = -0.3\%$.

So apparently one can calculate a^{pct} from any a^{ticks}, but not from the opposite direction if without rounding. Note that although here we only discuss the limit price, one can always append the volume v_t as the second term of the action.

Finally, we design the reward feedback. Like in real trading systems, we give the agent a reward only at the end of the last period. With the order schedule and the execution status of $t = 1, \ldots, T$, the reward is

$$R = \mathrm{D} \cdot \left[\sum_{t=1}^{T} v_t^* p_t^* - \left(\sum_{t=1}^{T} \tilde{v}_t \tilde{p}_t + \Delta_T \tilde{p}_{-1} \right) \right], \quad (2)$$

with p_t^* the market price of buying v_t^* in the t-th period, $\tilde{p}_t$ the agent's average execution price, Δ_T the final remained inventory, $\tilde{p}_{-1}$ the execution price of the final market order. The multiplier D denotes the trading direction, $\mathrm{D} = 1$ for buying and $\mathrm{D} = -1$ for selling. So using market orders yields a reward of 0.

3.2 Policy Optimization with Hybrid Action-spaces

In this part, we put forward our policy optimization method that balances generalization with continuous control and specialization with discrete control. We use a general stochastic policy optimization method, PPO [Schulman *et al.*, 2017], as our base learner.

There are two basic types of action-spaces, continuous and discrete, and the corresponding stochastic policies, as shown in Figure 1(a)(b). For the continuous action space $\mathcal{A}^{\text{pct}}$ to choose percentage changes in price, one can use a common parametric Gaussian policy with its mean μ and scale σ learned with a policy neural network, i.e.,

$$a_t^{\text{pct}} | s_t \sim \mathcal{N}(\mu(s_t), \sigma(s_t)).$$

For the discrete action space $\mathcal{A}^{\text{ticks}}$, a common choice is the Softmax policy, which randomly choose actions from a multinoulli categorical distribution

$$a_t^{\text{ticks}} | s_t \sim \text{Categorical}\big\{\text{Softmax}(\boldsymbol{f}_{1:m}(s_t))\big\},$$

where $\boldsymbol{f}_{1:m}(s_t)$ denotes the policy network's output logits for m candidate actions, and Categorical$\{\cdot\}$ is the categorical (multinoulli) distribution.

Although straight-forward, these policies are hard to optimize because the environment's state dynamics are too complex, the rewards are noisy and sparse, and the action space is too large especially for the discrete policy. To further restrict the search space, we propose a new family of policies: the *discretized Gaussian* policy, and its simplified version, the *Gaussian-and-Softmax* policy.

Discretized Gaussian Policies

Consider that we have a policy network that outputs the mean μ and scale σ of some Gaussian distribution $\mathcal{N}$, and a increasing series of predefined *locations* (candidate actions) $a_1^*, \ldots, a_m^*$ for discretization.

Let $a_0^* = -\infty$ and $a_{m+1}^* = +\infty$, for $1 \le k \le m$

$$d_k = \mathrm{P}\left(\frac{a_{k-1}^* + a_k^*}{2} < a < \frac{a_k^* + a_{k+1}^*}{2}\right) \quad (3)$$

$$= \Phi\left(\frac{a_k^* + a_{k+1}^* - 2\mu}{2\sigma}\right) - \Phi\left(\frac{a_{k-1}^* + a_k^* - 2\mu}{2\sigma}\right) \quad (4)$$

where $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\{-\frac{u^2}{2}\} du$ is the CDF of the standard normal distribution.

[1] Here "current price" refers to the latest executed price.

So now we can build a discretized Gaussian policy by using a categorical distribution to choose the percentage action, i.e.,

$$a_t^{\text{pct}}|s_t \sim \text{Categorical}\big\{d_1, \cdots, d_m\big\} \tag{5}$$

where the a^*s for calculating ds can be determined by the actual tick actions. In this case, such a policy can take the place of the original continuous Gaussian polices.

However, Eq.(4) involves an integral which is intractable for a feed-forward neural network. Therefore, we use uniform samples from $U(\frac{a_{k-1}+a_k}{2}, \frac{a_k+a_{k+1}}{2})$ and calculate the averaged Gaussian density so that the output action probability can be differentiable. I.e.,

$$\hat{d}_k = \frac{a_{k+1}-a_{k-1}}{2}\hat{\mathbb{E}}\Bigg[\frac{1}{\sigma\sqrt{2\pi}}\exp\left\{-\frac{(a-\mu)^2}{2\sigma^2}\right\} \Big| \, a \sim U(\frac{a_{k-1}+a_k}{2}, \frac{a_k+a_{k+1}}{2})\Bigg] \tag{6}$$

In this way, any Gaussian policy can be transformed into a trainable discrete policy, as demonstrated in Figure 1 (c). For readability, for a continuous policy $\pi(\cdot|s) = \mathcal{N}(\mu(s), \sigma(s))$, we write the discretized policy as Discretize$(\mathcal{N}(\mu(s), \sigma(s)))$.

Gaussian-and-Softmax Policy

A straight-forward simplification of the discretized Gaussian policy is to replace the sampling of Eq.(6) with a deterministic density function. So we can directly use a series of unnormalized logits $\boldsymbol{l} = (l_1, \ldots, l_m)$ to get a Softmax policy,

$$a_t^{\text{pct}}|s_t \sim \text{Categorical}\big\{\text{Softmax}(\boldsymbol{l})\big\}. \tag{7}$$

where $l_k = -\frac{(a_k-\mu)^2}{2\sigma^2}$. We write GSoftmax$(\mathcal{N}(\mu(s), \sigma(s)))$ to denote such a Gaussian-and-Softmax policy.

HALOP with Two Stages

From the derivation, now we can get a discretized policy from any continuous Gaussian policy. It motivates us to feel free to design a two-stage generalization-first-specialization-second method under the same policy family and network structure.

Stage 1: learn the most generalized knowledge without distinguishing different types of stocks. Therefore, we have the following configuration. Firstly, the input states in this stage only contains the standardized public state s_t^{pub} (by preprocessing all the prices and volumes to the same scale for different stocks), without the private execution status that reflects the characteristics of stocks. Second, the agent outputs the discretized percentage changes of price movements according to the realizable limit prices in order that the policy learned can be smoothly connected with the second stage. Formally, let p_t^{c} be the current price of a stock, and θ_1 be the parameters of the policy network in stage-1,

$$\mathcal{A}_{\text{S1}} = \left\{ a^{\text{pct}} \middle| a^{\text{pct}} = \frac{a^{\text{ticks}} \cdot \text{TickSize}}{p_t^{\text{c}}}, a^{\text{ticks}} \in \mathcal{A}^{\text{ticks}} \right\}, \tag{8}$$

$$a_{\text{S1},t}^{\text{pct}} \sim \pi_{\text{S1}}(\cdot|s_t^{\text{pub}}) = \text{Discretize}(\mathcal{N}(\mu_{\theta_1}(s_t^{\text{pub}}), \sigma_{\theta_1}(s_t^{\text{pub}}))), \tag{9}$$

$$a_{\text{S1},t}^{\text{ticks}} = a_{\text{S1},t}^{\text{pct}} \cdot p_t^{\text{c}} / \text{ TickSize}. \tag{10}$$

So the resulted output of the policy is a discrete action $a_{\text{S1},t}^{\text{ticks}}$ in the tick action space, which represents a general guess of the limit price. We would like to use this action to scope a subset of actions around it, as illustrated in Figure 1(d).

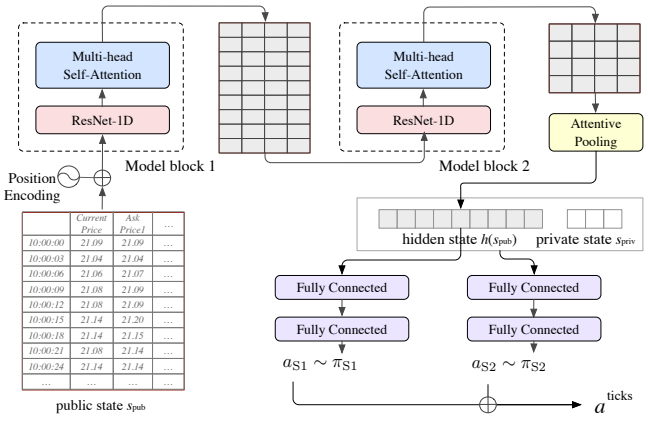


Figure 2: The architecture of HALOP.

Stage 2: After generalization, now for the specialization. We use the action offered by the policy of stage 1 as a medium point and search a fine-grained action around it in stage 2. We predefine a fixed-length window to scope the action subset of stage 2, i.e., consider a window-size of $2K+1$, the action subset of stage 2 is $\mathcal{A}_{\text{S2}} = \big\{-K, \ldots, 0, \ldots, K\big\}$. Therefore, as illustrated in Figure 1(e), for any stock with arbitrary price, the action-space in stage-2 is always $2K+1$ integers, encouraging the agent to focus on localized optimization. The state in stage 2 s_t includes the raw public state without standardization and the private states, which contains the absolute value of prices, volumes, and the agent's execution status, characterizing each single stock. For stabilized learning, we scale the prices and volumes by taking the logarithms of the absolute values. Formally, let θ_2 denote the parameters of the policy network in stage-2, we have

$$a_{\text{S2},t} \sim \pi_{\text{S2}}(\cdot|s_t) = \text{GSoftmax}(\mathcal{N}(\mu_{\theta_2}(s_t), \sigma_{\theta_2}(s_t))) \tag{11}$$

$$a_{\text{S2},t}^{\text{ticks}} = a_{\text{S1},t}^{\text{ticks}} + a_{\text{S2},t}. \tag{12}$$

3.3 Architectures and Implementation Details

In this part, we detail the choice of neural network architecture for learning from high-dimensional sequence data, as well as the learning details of PPO.

The overall network structure is shown in Figure 2. The input of the representation learning network is a sequence of high-dimensional tick snapshot data, i.e., the public states, with tens of real-valued features and hundreds of timesteps. First, we use two stacked neural blocks for sequence encoding, each block consists of a residual network with an 1-D convolutional neural network (1DCNN) to capture temporal patterns and shorten the sequence length, and a multi-head self-attention layer [Vaswani *et al.*, 2017] to learn the correlation among time steps. Finally, we get a dense representation $h(s)$ for the state by attentive pooling the output of the self-attention matrix.

Second, we use an actor-critic architecture [Schulman *et al.*, 2017; Pan *et al.*, 2019] for the policy optimization. For both stage-1 and stage-2, we use two fully-connected layers

as the policy and value networks. In stage-1, for generalization, the networks directly take the public $h(s_t^{\text{pub}})$ as input. In stage-2, we concatenate the private state to $h(s_t^{\text{pub}})$ to get a new dense vector as input. The two stages share the same representation learning network for better sample efficiency.

For policy optimization, we regard the order execution process of each (trading day, stock) pair as a single *episode*, which has T steps. Then, we group the episodes of all the stocks in each trading day as an $epoch$. For example, if there are 250 trade days and 300 stocks, then it groups into 250 epochs and 250×300 episodes. During training, we iterates between rollouts and updates in a batch RL manner. At each round, we randomly select one trading day from the training set, and run parallel rollout for all the episodes in that trading day. After finished, we perform the on-policy policy update with the collected data like the standard PPO algorithm [Schulman *et al.*, 2017].

4 Experiments

Our experiments are designed to investigate the following questions:

Q1. Can HALOP beat the market?

Q2. Can HALOP improve upon other volume-oriented order splitting strategies? Is HALOP stable and robust?

Q3. What benefits can we get from HALOP compared with vanilla continuous or discrete control?

Q4. Can HALOP address the trade-off between generalization and specialization better than other methods?

4.1 Experiment Setup

The Simulation Environment

We conduct experiments in simulation environments which is built over historical high-frequency transaction data of stocks of the CSI 300 index in the China A-shares. Specifically, we test all the methods in the buying direction and directly use the raw market information as the agent's state, which consists of the top 5 bid/ask prices and volumes, and the current (last) price. As described in the previous section, we use the data from January 2010 to December 2019 to build the training episodes, and use the data from January 2020 to June 2020 to build the testing episodes.

At each episode (i.e., for one stock on one trading day), our agent's execution mission starts at 10:00:00 (half an hour after opening) and ends at 14:30:00 (half an hour before close) which is an 180-minute trading hour. The time period is then split into $T = 90$ intervals evenly. At each step, the agents observes the market information and send an order to the market. We simulate a three second communication delay so as to make the experiment close to reality. For example, an order sent at 10:02:00 can be executed after 10:02:03. Moreover, to avoid potential large market impact, our environment does not execute any large order whose size is larger than $1/10$ of the agent's total inventory of the day. So on receiving an order with size v_t, the environment executes no more than $\min(v_t, 1/10)$ and cancels the rest of it, if there is.

Evaluation Metrics

Our evaluation metrics are designed around two core concepts in quantitative finance: the excess return and the risk.

We use the average excess return over TWAP-with-market-order as the **Return** metric. That is, the TWAP strategy using market orders always get a return of 0, and other methods get positive returns if it beats TWAP. The returns are displayed in basis points (bps, 1 bp equals to 0.01%).

Next, we compute the standard deviation of excess returns as a risk indicator, denoted as **Std**. Smaller standard deviation of returns means that the algorithm is more stable throughout different time and stocks.

With the computed Return and Std metrics, we calculate the t**-value** for paired student's t-test, to show the significance of the excess return. For any method, given the metric Return and Std, the t-value can be computed by

$$t\text{-value} = \frac{\text{Return}}{\text{Std}/\sqrt{n-1}}, \tag{13}$$

where n is the number of episodes. A larger t-value indicates more significant results. When n is large enough, $t > 3.29$ means that there is a improvement over the benchmark in confidence level 99.9%.

Finally, we need a more realistic metric that takes both return and safety constaints into consideration, which is our **PnL** (or Profit and Loss) for order execution. The profit is the excess return. For the loss, we take a regulation constraint into consideration: the cancellation rate for institutional investors cannot be greater than 50%. So when the cancellation rate exceeds 50%, we penalize the agent by -5 bps. So

$$\text{PnL} = \text{Return} - \sum_{i}^{n} 5 \cdot \mathbb{I}\big[\Delta_T + \sum_{t=1}^{T} v_t > 2\big], \tag{14}$$

with v_t defined in Eq.(1) and Δ_T the size of market order at the end of an episode.

4.2 Compared Methods

There are two categories of methods we compared: the price- and volume-oriented optimization methods. For the first part, we compare the following methods:

- **Market Order** which place market orders at all steps, meant to be executed as quickly as possible.
- **PPO-Gaussian** which is PPO [Schulman *et al.*, 2017] with a Gaussian policy (Figure 1(a)).
- **PPO-Softmax** which is PPO [Schulman *et al.*, 2017] with a Softmax policy (Figure 1(b)).
- **HALOP** is our proposed method with two stages.
- **HALOP Stage-1** is HALOP with only one stage (using the discretized Gaussian policy). It takes both public and private states as inputs. We test it to see whether it can improve upon the ordinary Gaussian policy.

For a fair comparison, all the RL algorithms are tested with the same network architecture (shown in Figure 2). The number of actions in HALOP stage-2 is set to $7 = 2 \times 3 + 1$.

To see how the agents perform under different volume schedule, we test the following volume-oriented methods:

- **TWAP** which evenly splits the order to T pieces and execute the same amount of shares at each step.

Method	TWAP				VWAP				ODP			
	Return	PnL	Std	t-value	Return	PnL	Std	t-value	Return	PnL	Std	t-value
Market Order	0.00	0.00	0.00	0.0	0.14	0.14	8.72	3.1	1.64	0.92	65.34	4.8
PPO-Gaussian	1.41	1.33	4.61	58.7	1.38	1.31	8.94	29.7	2.20	1.12	64.55	6.5
PPO-Softmax	2.12	-1.46	35.88	11.3	2.36	-1.17	35.47	12.8	3.34	-0.53	66.73	9.6
HALOP Stage-1	3.69	1.46	17.05	41.5	3.87	1.68	17.51	42.4	4.51	1.45	63.67	13.6
HALOP	**4.41**	**2.98**	6.87	123.2	**4.52**	**3.12**	9.24	93.8	**5.04**	**2.46**	63.42	15.3

Table 1: Main results.

- **VWAP** which sends orders in proportion to an estimated volume ratio of each time period of the day, which is estimated from the previous 21 trading days.
- **OPD** [Fang *et al.*, 2021] which leverages RL with policy distillation to determine the size of each sub-order.

4.3 Main Experimental Results

Table **??** reports the overall performance.

First, to answer question **Q1**, it is clear that our HALOP method can indeed outperform the market orders with a large margin. When using the basic TWAP, HALOP beats Market Order with 4.41 bps and a PnL of 2.98, with a t-value of 123.2 which proves the significance of improvements.

Second, to answer **Q2**, we would like to see how HALOP performs under other volume allocation methods. We firstly see that the superiority of HALOP against Market Order still holds with either TWAP, VWAP or ODP. Also, surprisingly, we find that when using HALOP, the Std of ODP decreased comparing to the original ODP with market order, indicating that HALOP can even stabilize the order execution process in certain circumstances.

Third, to answer **Q3**, we need to compare HALOP with ordinary PPOs as well as the lite version of HALOP with only stage-1. We find that though PPO-Gaussian and PPO-Softmax all yields positive excess returns, the improvements are not as large as that of HALOP. Moreover, while PPO-Softmax gets a higher excess-return than PPO-Gaussian, it has negative PnL and very large Std. It indicates that although the Softmax policy is capable of finding the best action, it is unstable and unreliable. We also see that HALOP Stage-1 yields significantly better result than the ordinary PPOs, showing its ability of taking the advantages of both Gaussian policy and discrete control. Finally, the comparison between HALOP and HALOP Stage-1 shows that the specialization in stage-2 indeed helps improve both excess return and stability.

4.4 Grouping Study

We further design a grouping study to see whether HALOP address the trade-off between generalization and specialization well. We group the test episodes by stock close prices: "low-priced" for price less than 10.00 CNY, "medium-priced" for price between 10.00-50.00 CNY, and "high-priced" for larger than 50.00 CNY.

As shown in Figure 3, we find PPO-Gaussian more in favor of high-priced stocks than low-priced stocks. Next, we observe that HALOP Stage-1 has a strong ability of generalization in all the groups: the excess returns are all over 3.5. Further, the two-stage HALOP further improves the results especially for low-priced stocks, which indicates that the fine-grained discrete control in stage-2 indeed achieves better specialization for different types of stocks. Specifically, for penny stocks which are known to have less liquidity, the discretization in HALOP Stage-2 is shown to be more beneficial.

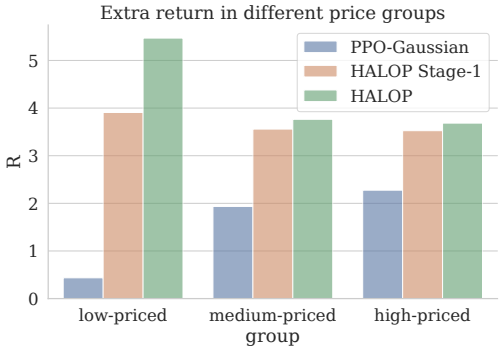


Figure 3: Results of grouping study

5 Conclusion

This paper focuses on optimal execution with limit orders by setting better limit prices. We propose a novel reinforcement solution, Hybrid Action-Space Order Placement, to address the generalization-specialization trade-off by combining both advantages of continuous and discrete control. The proposed method has two stages: in stage-1, the agent selects a target percentage change in price with a discretized Gaussian policy and scopes a subset of discrete actions thereby; in stage-2, the agent uses a fine-grained Gaussian-and-Softmax policy to select a tick-based action. Extensive simulation-based experiments show that our method can beat the market and improve upon other volume-oriented order splitting strategies. It shows stable and robust improvements comparing with vanilla continuous or discrete control methods. Specifically, the discretized Gaussian policy of stage-1 helps improve the generalization ability, and meanwhile in stage-2 the agent specializes in policy for different types of stocks especially for low-priced stocks. In a broader sense, we believe that HALOP offers a reference of applying RL in real-world applications where the trade-off between generalization and specialization should be taken into consideration.

References

[Almgren and Chriss, 2001] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2001.

[Bertsimas and Lo, 1998] Dimitris Bertsimas and Andrew W Lo. Optimal control of execution costs. *Journal of Financial Markets*, 1(1):1–50, 1998.

[Cartea *et al.*, 2015] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. *Algorithmic and high-frequency trading*. Cambridge University Press, 2015.

[Fang *et al.*, 2021] Yuchen Fang, Kan Ren, Weiqing Liu, Dong Zhou, Weinan Zhang, Jiang Bian, Yong Yu, and Tie-Yan Liu. Universal trading for order execution with oracle policy distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 107–115, 2021.

[Hendricks and Wilcox, 2014] Dieter Hendricks and Diane Wilcox. A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. In *2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*, pages 457–464. IEEE, 2014.

[Huberman and Stanzl, 2000] G. Huberman and W. Stanzl. Optimal liquidity trading. *Yale School of Management Working Papers*, 9(2):165–200, 2000.

[Kakade *et al.*, 2004] Sham M Kakade, Michael Kearns, Yishay Mansour, and Luis E Ortiz. Competitive algorithms for vwap and limit order trading. In *Proceedings of the 5th ACM conference on Electronic commerce*, pages 189–198, 2004.

[Lin and Beling, 2019] Siyu Lin and AP Beling. Optimal liquidation with deep reinforcement learning. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019) Deep Reinforcement Learning Workshop, Vancouver, Canada*, 2019.

[Lin and Beling, 2020] Siyu Lin and Peter A Beling. An end-to-end optimal trade execution framework based on proximal policy optimization. In *IJCAI*, pages 4548–4554, 2020.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[Nevmyvaka *et al.*, 2006] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680, 2006.

[Ning *et al.*, 2018] Brian Ning, Franco Ho Ting Lin, and Sebastian Jaimungal. Double deep q-learning for optimal execution. *arXiv preprint arXiv:1812.06600*, 2018.

[Pan *et al.*, 2019] Feiyang Pan, Qingpeng Cai, An-Xiang Zeng, Chun-Xiang Pan, Qing Da, Hualin He, Qing He, and Pingzhong Tang. Policy optimization with model-based explorations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4675–4682, 2019.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[Watkins and Dayan, 1992] Cjch Watkins and P. Dayan. Q-learning. In *Machine Learning*, 1992.