Towards Generalizable Reinforcement Learning for Trade Execution

Chuheng Zhang[1] , **Yitong Duan**[2] , **Xiaoyu Chen**[2] , **Jianyu Chen**[2] , **Jian Li**[2] and **Li Zhao**[1]
[1]Microsoft Research
[2]IIIS, Tsinghua University
zhangchuheng123@live.com, {dyt19, chen-xy21, lijian83}@mails.tsinghua.edu.cn,
jianyuchen@tsinghua.edu.cn, lizo@microsoft.com

Abstract

Optimized trade execution is to sell (or buy) a given amount of assets in a given time with the lowest possible trading cost. Recently, reinforcement learning (RL) has been applied to optimized trade execution to learn smarter policies from market data. However, we find that many existing RL methods exhibit considerable overfitting which prevents them from real deployment. In this paper, we provide an extensive study on the overfitting problem in optimized trade execution. First, we model the optimized trade execution as offline RL with dynamic context (ORDC), where the context represents market variables that cannot be influenced by the trading policy and are collected in an offline manner. Under this framework, we derive the generalization bound and find that the overfitting issue is caused by large context space and limited context samples in the offline setting. Accordingly, we propose to learn compact representations for context to address the overfitting problem , either by leveraging prior knowledge or in an end-to-end manner. To evaluate our algorithms, we also implement a carefully designed simulator based on historical limit order book (LOB) data to provide a high-fidelity benchmark for different algorithms. Our experiments on the high-fidelity simulator demonstrate that our algorithms can effectively alleviate overfitting and achieve better performance.

1 Introduction

Nowadays, brokerage firms are required to execute orders on behalf of their clients (e.g., retail or institutional investors) to ensure execution quality. Optimized trade execution, whose objective is to minimize the execution cost of trading a certain amount of shares within a specified period, is an important task towards better execution quality. In modern financial markets, most of the transactions are conducted through electronic trading. Therefore, developing a smart agent for optimized trade execution in electronic markets is a critical problem in the financial industry.

Traditional solutions for this problem [Almgren and Chriss, 2001; Guéant *et al.*, 2012; Bulthuis *et al.*, 2017] usually make strong assumptions on the price or transaction dynamics and therefore do not apply to real scenarios. Moreover, these strategies are static (i.e., determined before the start of the trading) and therefore unable to adapt to the real-time market. Recently, RL-based methods have been developed to learn a more adaptive agent from market data [Fang *et al.*, 2021; Ning *et al.*, 2018; Lin and Beling, 2020]. However, we find that existing methods suffer from considerable overfitting. As shown later in Figure 1 (right), trained models are prone to memorize the history instead of learning generalizable policies.

To better analyze overfitting in trade execution, we propose a framework called Offline Reinforcement learning with Dynamic Context (ORDC) to model the problem. This framework highlights the difficulty in generalization for the trade execution task. In trade execution, part of the observation (which we call *context*) evolves independently of the agent's action, and the simulator is based on a dataset that contains a finite number of context sequences (e.g., historical price sequences). The number of context sequences is usually limited and does not increase w.r.t. the number simulation steps at training time. Therefore, the agent is prone to memorize the training context sequences, and perform not well on testing context sequences. We highlight the difficulty in generalization under this setting theoretically and show that limited context sequences lead to bad generalization. This explains why generalization is hard for ORDC. The offline nature does not receive much attention in previous RL applications for trade execution that usually employ off-the-shelf online RL methods with data-driven simulation.

Since it is usually hard to increase the number of sampled context sequences for training in practice, we find another way to address the overfitting problem. The theoretical analysis also indicates that larger context space leads to worse generalization under the same number of samples. Motivated by the analysis, we propose to aggregate the context space by learning a compact context representation for better generalization. This is effective for trade execution where the context usually contains more information than needed, but only a small amount of the underlying information is helpful for decision-making. Moreover, we design a simplified trade execution task that motivates us to learn a compact context representation that is predictive of the statistics on future contexts. Therefore, we propose to use the prediction of future statistics as the context representation. We propose two algorithms: **CASH** (**C**ontext **A**ggregate with **H**and-crafted **S**tatistics) which can learn interpretable models, and **CATE** (**C**ontext **A**ggregate with **E**nd-to-end **T**raining) which does not require domain knowledge.

In the experiment, we first implement a high-fidelity and open-source simulator for trade execution to provide a uniform backtest for different methods. With this simulator, we find that previous state-of-the-art algorithms suffer from overfitting and our algorithm outperforms these baselines due to better generalization.

The contributions of this paper are as follows:

- (Section 3) We propose the ORDC framework and provide theoretical analysis to highlight the difficulty in generalization for the trade execution task.
- (Section 4) We propose two algorithms to learn generalizable representations for trade execution. One is interpretable with the help of human prior and the other learns in an end-to-end manner.
- (Section 5) We implement an open-source, high-fidelity, and flexible simulator to reliably compare different trade execution algorithms. With this simulator, we show that our models learn more generalizable policies and outperform previous methods.

2 Related Work

Most existing papers on RL generalization study under the contextual MDP setting where the agent is trained and evaluated on different sets of configurations [Zhang *et al.*, 2018b; Zhang *et al.*, 2018a; Packer *et al.*, 2018] or procedurally generated environments [Cobbe *et al.*, 2019; Cobbe *et al.*, 2020; Song *et al.*, 2019; Wang *et al.*, 2020]. ORDC is different from their settings in that 1) the context changes in each time step and affects both the reward and the transition; 2) the context sequence is highly stochastic and pre-collected with limited volume. These properties contribute to the difficulty in estimating the value function or evaluating the policy and thus exacerbate overfitting. A recent survey [Kirk *et al.*, 2021] points out that benchmarking RL algorithms with popular procedurally generated environments is not enough and RL generalization in other settings (e.g., the offline setting) is valuable and under-explored.

Not only limited to trade execution, the structure of ORDC is common for many industrial RL application scenarios such as video stream control [Mao *et al.*, 2017], inventory management [Oroojlooyjadid *et al.*, 2022], ride-sharing [Shen *et al.*, 2020], cellular network control [Dietterich *et al.*, 2018], etc. Although ORDC emphasizes the offline nature, it is also different from the canonical offline RL setting. The agent is trained to avoid encountering unseen states in offline RL, whereas the agent in ORDC is evaluated on unseen context sequences which cannot be avoided. Therefore, off-the-shelf offline RL algorithms do not apply to our setting and a more adaptive algorithm is needed. Many previous solutions for trade execution use online RL algorithms to learn a policy from the interaction with the data-driven simulator. However, unlike online RL settings where the testing and training environments are the same, ORDC tests the agent on unseen context sequences which brings in difficulty in generalization. IDSD [Shahamiri, 2008] and input-driven MDP [Mao *et al.*, 2018] are similar to ORDC in that they model the uncontrollable part in observation. However, they focus on the online setting and do not model the existence of a smaller latent context space (see section 3).

3 Why Generalization is Difficult for Trade Execution?

In this section, we first briefly introduce the trade execution problem. Then, we introduce Offline RL with Dynamic Context (ORDC) which models how RL is used to solve the problem. At last, we provide theoretical analysis for the ORDC model to highlight the difficulty in generalization for these RL applications.

3.1 Trade Execution

Modern electronic markets match the buyers and sellers with the limit order book (LOB), which is a collection of outstanding orders, each of which specifies the direction (i.e., buy or sell), the price, and the volume. The traders can trade via two types of orders: market orders (MOs) and limit orders (LOs). An MO is executed immediately but may suffer from a large trading cost (e.g., due to crossing the spread or temporary market impact [Almgren and Chriss, 1999]), whereas an LO can provide the trader with a better price but at the risk of non-execution. See appendix for more details on LOB. Moreover, the price fluctuation or trend can also affect the trading cost. Optimized trade execution aims to buy/sell a given amount of assets in a given time period at a trading cost as low as possible. For simplicity, we only consider liquidating (or selling) the asset.

Previously, different methods [Nevmyvaka *et al.*, 2006; Lin and Beling, 2020; Fang *et al.*, 2021] are proposed to apply RL to trade execution, but they follow a similar procedure: The agent learns based on interactions with a data-driven simulator. The dataset contains the information collected from the real market and is used to determine simulated transitions. Therefore, it can be regarded as an offline RL setting. The observation of the agent can be divided into the market variable (e.g., the LOB snapshot) and the private variable (e.g., remaining time and inventory). The market variable is usually high-dimensional and incorporates different forms of information to represent the noisy and partially observable market.

3.2 Offline RL with Dynamic Context

To model the problem structure when applying RL to trade execution, we introduce the ORDC model. ORDC is a tuple $(\mathcal{X}, \mathcal{C}, \mathcal{S}, \phi, P, r, \gamma, \mathcal{D})$ specifying the latent context space $\mathcal{X}$, the context space $\mathcal{C}$, the state space $\mathcal{S}$, the unknown context decoding function $\phi : \mathcal{C} \rightarrow \mathcal{X}$, the transition dynamics $P(x', s'|x, s, a) = P_x(x'|x)P_s(s'|x, s, a)$, the reward $r(x, s, a)$, the discount factor γ and the offline context dataset $\mathcal{D}$ that contains context sequences.

In trade execution, the market variable serves as the context, and the private variable serves as the state. The context evolves independently and is not affected by the action or the state. However, the context is important since it influences the

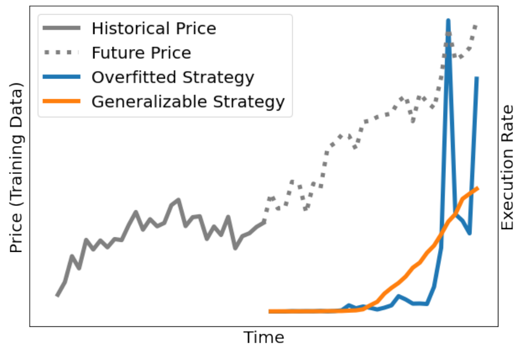

Figure 1: The blue line: In trade execution, the deep learning agent can memorize and overfit to the context sequence used for training (the gray line) and liquidate most of the inventory on the highest price. The orange line: A generalizable agent should output a smooth policy considering the stochasticity of future context. See the corresponding experiment setting in Section 5.1.

reward collected by the agent and the dynamics of the state. The context $c \in \mathcal{C}$ is usually high-dimensional and corresponds to a more compact latent context $x \in \mathcal{X}$ (e.g., the key information for making trading decisions). Specifically, they have the *block structure* [Du *et al.*, 2019]: Each context $c \in \mathcal{C}$ uniquely determines its generating latent context $x \in \mathcal{X}$ with the unknown mapping $\phi : \mathcal{C} \to \mathcal{X}$.

Given a policy $\pi : \mathcal{C} \times \mathcal{S} \to \Delta^{\mathcal{A}}$, the Q-function is defined as $Q^\pi(c, s, a) := \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(x_t, s_t, a_t) | x_0 = \phi(c), s_0 = s, a_0 = a, \pi]$, where $a_t \sim \pi(\cdot|c_t, s_t)$ and x_t, s_t transits following the dynamics P for all $t \geq 1$. The agent learns from interactions with a data-driven simulator based on the offline dataset $\mathcal{D}$ and outputs a policy π that maximizes $J(\pi) = \mathbb{E}[Q^\pi(c, s, a)|(c, s) \sim P_0, a \sim \pi(\cdot|c, s)]$ where P_0 is the initial context/state probability.

3.3 Theoretical Analysis for Generalization under ORDC

With a slight abuse of notation, we can write the dynamics as $P(c', s'|c, s, a) = P_c(c'|c)P_s(s'|c, s, a)$ owning to the block structure. In many real instances of ORDC, simulation is cheap and thus P_s can be accurately estimated from a large number of interactions with the simulator. Moreover, P_s (e.g., the rules to match the orders) is usually simple, whereas P_c (e.g., involving market dynamics) is complex and hard to estimate. Therefore, we further assume P_s is known. The following sample complexity lower bound highlights the intrinsic difficulty in generalization under ORDC. (Notice that the sample complexity indicates how many samples are sufficient to ensure a small generalization gap.)

Assumption 1 (Regularity)**.** *$\mathcal{C}$, $\mathcal{X}$, $\mathcal{S}$, and $\mathcal{A}$ are discrete and the immediate reward $r(x, s, a) \in [0, 1], \forall x, s, a$.*

Theorem 1. *Under Assumption 1, there exists a class of ORDC models $\mathbb{M} = \{M_1, \cdots, M_m\}$ such that any algorithm A needs at least $T = \Omega\left(\frac{|\mathcal{C}| \log(|\mathcal{C}|/\delta)}{(1-\gamma)^3 \epsilon^2}\right)$ context samples to learn a value function $Q^A \in \mathbb{R}^{\mathcal{C} \times \mathcal{S} \times \mathcal{A}}$ such that $\|Q^* - Q^A\|_\infty \leq \epsilon$ with probability at least $1 - \delta$ for all $M \in \mathbb{M}$, where Q^* is the optimal action value function.*

We provide the proof in appendix. In the proof, we construct a class of ORDC models where the contexts can be divided into a small number of categories. However, without further knowledge on how to aggregate the contexts, the algorithm still needs a large number of samples (i.e., $\tilde{O}(|\mathcal{C}|)$) to learn a generalizable policy. We will later show that the sample complexity can be improved when the context aggregation is known.

The theorem indicates that the estimated value function can overfit to limited context sequences when the context space is large or the underlying context dynamics is complex. Actually, this is the case for real trade execution tasks. First, the context space is large since people usually incorporate many market indicators as the context to reflect the market more comprehensively. Second, the context dynamics is complex and highly stochastic since it is driven by various market participants, news, economics, etc. Moreover, we find that function approximation does not effectively improve generalization since deep learning models can also suffer from such overfitting. To illustrate this, we present an overfitted strategy and a generalizable strategy in Figure 1. The overfitted strategy results from a deep learning model trained using a standard RL algorithm, and the generalizable strategy results from a similar training but with a technique that aggregates the context (see Section 5.1 for details). We can see that a standard deep RL model can memorize the highest price in the training context (price) sequence and learn an aggressive policy that liquidates nearly all the stocks on that price.

4 Towards Better Generalization

Motivated by the theoretical analysis, we first show that aggregating the context can improve the generalization theoretically. Then, we introduce two practical algorithms for trade execution: 1) **CASH** (**C**ontext **A**ggregate with **H**and-crafted **S**tatistics) , and 2) **CATE**(**C**ontext **A**ggregate with **E**nd-to-end **T**raining).

4.1 Context Aggregation

In ORDC, the agent may overfit to limited context sequences in the dataset. We observe that, by resorting to the context decoding function $\phi : \mathcal{C} \to \mathcal{X}$ that maps the high-dimensional context into the latent context, we can obtain a more generalizable agent.

Theorem 2. *With the access to ϕ and a generative model to collect context samples for $\mathcal{D}$, there exists an algorithm that learns a value function $\hat{Q}$ such that $\|Q^* - \hat{Q}\|_\infty \leq \epsilon$ as long as the context transitions in $|\mathcal{D}|$ is larger than $D = O\left(\frac{|\mathcal{X}|^2 \log(|\mathcal{X}|/\delta)}{(1-\gamma)^4 \epsilon^2}\right)$ with probability at least $1 - \delta$.*

We provide the proof of Theorem 2 in appendix. [1] Notice that $\mathcal{X}$ is considered to have a much smaller cardinality than $|\mathcal{C}|$. This indicates that we can learn a good policy with much fewer context samples when ϕ is available. However,

[1]Compared with Theorem 1, the additional dependency on the cardinality of the (latent) context space may be improved using more involved analysis (e.g., using a Bernstein style inequality).

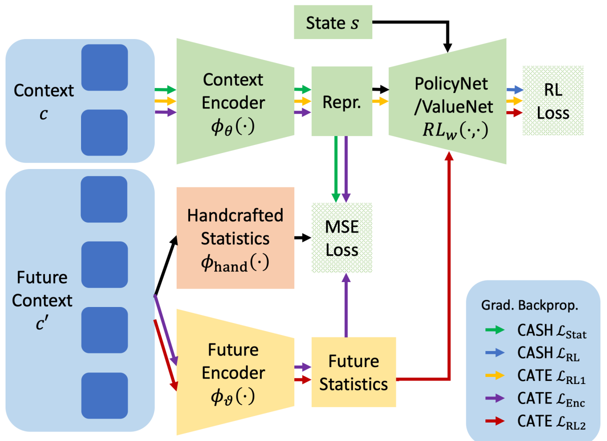
State s
Context c
Context Encoder $\phi_\theta(\cdot)$
Repr.
PolicyNet /ValueNet $RL_w(\cdot,\cdot)$
RL Loss
Handcrafted Statistics $\phi_{\text{hand}}(\cdot)$
MSE Loss
Future Context c'
Future Encoder $\phi_\vartheta(\cdot)$
Future Statistics
Grad. Backprop.
CASH $\mathcal{L}_{\text{Stat}}$
CASH $\mathcal{L}_{\text{RL}}$
CATE $\mathcal{L}_{\text{RL1}}$
CATE $\mathcal{L}_{\text{Enc}}$
CATE $\mathcal{L}_{\text{RL2}}$

Figure 2: The architecture of variational autoencoder

the mapping ϕ is not provided in many real scenarios. Nevertheless, we still hope to improve the generalization of the model by finding a mapping that can effectively aggregate the high-dimensional context. Next, we propose two algorithms to approximate the mapping either using the domain knowledge or an end-to-end training scheme.

4.2 Practical Algorithms

With a simulator that replays the historical context sequences, we can use the standard online RL algorithm by treating it as a regular MDP (where the observation contains the context as well as the state). Additionally, we consider the risk of overfitting highlighted in the analysis on ORDC and propose to train a context encoder to approximate the mapping ϕ.

In our algorithms, we guide the learning of the context encoder with the statistics extracted from future contexts. The reasons are as follows: The key challenge in trade execution is to handle the uncertainty of the future since the solution for the task would be easy if we knew the future, e.g., liquidating on the known highest price. In this sense, compared with traditional methods that do not use context and output static policies, using context gives us a good indication of the future and enables dynamic adaptation. Therefore, the information extracted from future contexts is important to guide the training of the context encoder. However, there is spurious noise in the future contexts that is not predictable from the current context. Therefore, we hope to use stable statistics (i.e., with a stable correlation with the optimal decision, see [Arjovsky *et al.*, 2019]) that can extract predictable as well as generalizable information from future contexts. For example, crude price movement and volatility in the future are rather predictable, whereas the specific time point in the future when the price is highest is not generalizable. Moreover, the design of our algorithms is also motivated by the experiments on a simplified trade execution task (See Section 5.1), which indicates that guiding the context encoder with the statistics on future contexts is a simple yet effective method.

4.3 CASH: Context Aggregation with Hand-crafted Statistics

CASH and CATE are based on the training of a standard DRL model $RL_w(\phi_\theta(c), s)$ that receives a context c and a state s and outputs actions or values. We use θ and w to denote the parameters in the context encoder and the policy/value network respectively. CASH uses hand-crafted statistics to guide the learning of the context encoder. We present the diagram of CASH in Figure 2. In the pre-training phase, we first train a context encoder that tries to predict the labels extracted using hand-crafted statistics from future contexts. Specifically, we train the encoder $\phi_\theta(\cdot)$ with the loss $\mathcal{L}_{\text{Stat}}(\theta) := (\phi_\theta(c) - \phi_{\text{hand}}(c'))^2$ where c' is the future context following the current context c and ϕ_{hand} extracts hand-crafted statistics. The loss is a mean-squared error between the context representation and the generated statistics. In the training phase, we fix the context encoder $\phi_\theta(\cdot)$ and train the policy $RL_w(\cdot,\cdot)$ using standard RL algorithms. We denote the loss function of the RL algorithm as $\mathcal{L}_{\text{RL}}(w)$ which can be the TD error for value-based RL algorithms (e.g., DQN) or the negative policy performance estimation in policy gradient algorithms (e.g., PPO). These losses are estimated and optimized based on the transition samples collected from the data-driven simulator. Specifically, the context c and future context c' are collected by replaying the historical data; the state s and the reward are calculated by the simulator.

In our experiment, we design statistics of the future contexts $\phi_{\text{hand}}(\cdot)$ based on the observation that the task would be simple if the information about the future price trend and spread is discovered. We use the following hand-crafted statistics: 1) The difference between the average future twap (time-weighted average price) and the current twap, which indicates the trend; 2) The difference between the maximum/minimum future twap and the current twap, which indicates whether the current price is a peak/bottom; 3) The volatility of the future twap, which is related to risk control; 4) The standard deviation of future spreads and the difference between the average/maximum/minimum of future spreads, which are related to the temporary market impact.

4.4 CATE: Context Aggregation with End-to-end Training

Designing effective statistics for the context encoder to predict requires expertise in the specific domain, which is unavailable in many real scenarios. Therefore, we propose CATE that learns to generate such statistics via a future (context) encoder $\phi_\vartheta(\cdot)$ where ϑ is the trainable parameter. We present the algorithm in Figure 2. The algorithm learns a context encoder (that outputs the context representation), a future encoder (that outputs the future statistics), and a policy/value network. These components are trained simultaneously with the following loss in an in an end-to-end manner:

$$\mathcal{L}(\theta,\vartheta,w) = \sum_{\text{transitions}} \mathcal{L}_{\text{Enc}}(\theta,\vartheta) + \mathcal{L}_{\text{RL1}}(\theta,w) + \mathcal{L}_{\text{RL2}}(\vartheta,w)$$

$$\begin{aligned} \text{where } & \mathcal{L}_{\text{Enc}}(\theta,\vartheta) := (\phi_\theta(c) - \phi_\vartheta(c'))^2 \\ & \mathcal{L}_{\text{RL1}}(\theta,w) \text{ optimizes the RL model } RL_w(\phi_\theta(c), s) \\ & \mathcal{L}_{\text{RL2}}(\vartheta,w) \text{ optimizes the RL model } RL_w(\phi_\vartheta(c'), s) \end{aligned}$$

The first term $\mathcal{L}_{\text{Enc}}$ is used not only to train the context encoder but also encourage the future encoder to generate predictable statistics. The second term $\mathcal{L}_{\text{RL1}}$ optimizes the base RL model (i.e., the context encoder and the value/policy network). The third term $\mathcal{L}_{\text{RL2}}$ encourages the future encoder to generate statistics that are helpful for decision making and therefore incentivizes informative future statistics. Similar to CASH, this loss is calculated based on the transition samples collected from the data-driven simulator.

5 Experiments

In this section, we first conduct a simplified task of trade execution, to illustrate the overfitting in vanilla RL methods and the effectiveness of context aggregation for generalization. Then, based on our market simulator, we compare our algorithms with other existing trade execution methods in the real stock market data. We provide the source code in https://github.com/zhangchuheng123/RL4Execution.

5.1 Experiments on Simplified Trade Execution Task

In this experiment, we introduce a simplified trade execution task to study the overfitting problem with a context dataset and possible solutions towards better generalization. In this simplified task, all transactions are executed on a single price process (i.e., without ask/bid price) following the Brownian motion. Formally, the dynamic of price process is $p_{t+1} = p_t + \Delta p_t, \Delta p_t = \alpha + \sigma \xi_t$, where p_t is the price at the t-th step, $\xi_t \sim \mathcal{N}(0, 1)$ is a random variable which follows the the standard Gaussian distribution independently at each step t, α and σ denote drift and volatility which are two statistical parameters of process. The task is to learn an agent that can give a execution strategy based on the observation of price changes over the past 30 time steps.

The objective is to reduce the trading cost, and therefore we set the negative discounted trading cost as the reward (i.e., the gap between the average discounted execution price and the average discounted market price).

This task is a simplified ORDC task, which focuses on 1) the existence of a mapping between the high-dimensional context c and the latent context $x = (\alpha, \sigma)$, and 2) training with a limited context dataset. In the following experiments, we use DDPG [Silver *et al.*, 2014] as the base RL algorithm. We design several methods to solve the task and observe the corresponding generalization ability of the learned agents. We present the experiment results in Table 1 and analyze the result of each model as follows:

Base. We can first observe the performance of the base RL model. We can see that it performs well when the sample data is sufficient, but its performance degenerates quickly when the data volume decreases.

Bottleneck. When the data is noisy and limited, deep learning models with high capacity are able to memorize the samples in the training set. A natural idea is to limit the model capacity with a representation bottleneck (i.e., learning a low-dimensional representation). With the prior knowledge that the whole price process can be represented by two parameters (i.e., drift α and volatility σ), so we set the representation to be a two-dimensional vector. However, we observe that an end-to-end training process for a model with bottleneck does not result in a good performance. Additionally, we observe that the training process is highly unstable.

CATE. In this model, we consider an encoder-decoder architecture to learn the representation with the others remaining the same as *Bottleneck*. Specifically, the encoder generates a two-dimensional representation from the past context, and the decoder tries to predict the future context sequence based on the representation (the decoder that reconstructs the past context sequence results in similar performance). The decoder receives a two-dimensional representation generated by the encoder and tries to predict the future context sequence. We observe that this model results in relatively good performance even when the data is highly limited. However, the two-dimensional representation may not only embed the estimated statistical parameters (i.e., (σ, α)) but also overfit the spurious noise in the data (i.e., ξ_ts).

CASH. In this model, we consider using a separate training signal to supervise the learning of the representation with other architectures remaining the same as *Bottleneck*. The loss function to train the encoder is a mean-squared-error w.r.t. a two-dimensional hand-crafted target vector, which is the estimate of $(\hat{\sigma}, \hat{\alpha})$. We observe that this model achieves superior performance even when the data is highly scarce. Moreover, the generalization gap is only half of that in the previous model, which may benefit from the fact that this model avoids fitting the spurious noise.

We also present the strategies learned by *Base* and *CASH* with 1k data in Figure 1. We can observe that the strategy learned by *Base* (cf. overfitted strategy in Figure 1) presents sharp peaks resulting from overfitting the training data. In contrast, the strategy learned by *CASH* (cf. generalizable strategy in Figure 1) is smooth which indicates that the agent is more generalizable.

Through the experiments on this simplified trade execution task, we have several observations: 1) Overfitting can easily occur for a deep RL model even in a setting simpler than the ORDC model. 2) Simply regularizing the capacity of the representation does not lead to better performance or generalization. 3) Reconstruction/Prediction-based encoder training combined with limited representation capacity can achieve good performance. 4) With carefully designed target features, we can prevent the encoder from fitting spurious noise and further improve generalization.

5.2 Experiments with High-Fidelity Simulation

Simulated Environment. To reduce the gap between simulation and the real-world environment and provide a reliable benchmark for different algorithms, we build an open source, high fidelity, and flexible market simulator for trade execution. Compared with previous simulators that are based on bar-level simulation [Fang *et al.*, 2021], our simulator is based on LOB-level data and thus has higher fidelity. Specifically, our simulator considers the temporary market impact, time delay, and second-level snapshot-by-snapshot order book reconstruction to minimize the sim-to-real gap. Moreover, our simulator can easily adapt for different designs

		Reward (train)		Reward (eval)		Gap	
Data volume	Model	mean	std	mean	std	mean	std
100k	*Base*	1.8559	0.0379	**1.8291**	0.0227	0.0268	0.0317
	Bottleneck	-0.0024	0.0035	-0.0022	0.0162	-0.0003	0.0150
	CATE	1.7143	0.0281	1.7004	0.0265	0.0139	0.0252
	CASH	1.8016	0.0246	1.8028	0.0187	-0.0013	0.0321
10k	*Base*	2.5389	0.0197	1.6047	0.0266	0.9341	0.0346
	Bottleneck	0.0059	0.0131	-0.0006	0.0013	0.0065	0.0144
	CATE	1.7743	0.0261	1.7335	0.0536	0.0408	0.0471
	CASH	1.8329	0.0300	**1.8083**	0.0304	0.0246	0.0544
1k	*Base*	4.0340	0.0659	1.4083	0.0748	2.6258	0.1259
	Bottleneck	0.3202	0.8233	0.4142	0.8833	-0.0940	0.1430
	CATE	2.0324	0.1823	1.6007	0.0591	0.4317	0.1736
	CASH	2.1578	0.1219	**1.9557**	0.0555	0.2021	0.1188

Table 1: The performance of different models on the simplified trade execution task. The models are evaluated over five random seeds. Reward (train/eval) represents the negative trading cost on the training/evaluation set.

Algorithm	Training	Validation	Testing	Gap
TWAP	-	-	14.0984 (2.1545)	-
Momentum	-	-	12.2530 (0.6151)	-
Tuned DQN	2.0382 (1.7684)	5.8134 (2.1032)	5.9240 (3.2986)	3.8858
[Nevmyvaka *et al.*, 2006]	3.0781 (5.2447)	8.8698 (1.5701)	9.1223 (1.0554)	6.0441
[Ning *et al.*, 2018]	7.3248 (5.1059)	10.3971 (2.0066)	9.4051 (2.6524)	2.0804
[Lin and Beling, 2020]	5.8778 (7.0791)	10.7000 (0.7024)	12.7116 (1.1514)	6.8338
Tuned DQN + CASH	2.2269 (2.0798)	3.7992 (1.4612)	3.4250 (1.9052)	**1.1981**
Tuned DQN + CATE	-2.8774 (1.7019)	-1.8431 (1.2983)	**0.0075** (1.6920)	2.8849
Tuned PPO	-0.9505 (2.2439)	2.1132 (0.2497)	2.7575 (1.2070)	3.7079
[Dabérius *et al.*, 2019]	7.6944 (11.3490)	9.2893 (1.9000)	12.3166 (2.8627)	4.6222
[Lin and Beling, 2021]	5.2697 (7.4173)	8.1153 (0.8894)	9.4807 (1.9686)	4.2110
[Fang *et al.*, 2021]	-4.9090 (16.1474)	10.1338 (4.0843)	11.8739 (5.0948)	16.7829
Tuned PPO + CASH	-4.6504 (0.4916)	-3.9351 (0.2810)	-4.5760 (0.2062)	**0.0744**
Tuned PPO + CATE	-5.0364 (0.8104)	-3.8797 (0.1103)	**-4.9068** (0.3015)	0.1296

Table 2: The trading cost (bp=10^{-4}) of different algorithms. The validation set is used for hyperparameter tuning. The numbers are the average mean (std.) trading cost in the last 100 evaluations of the total 1000 evaluations over five different random seeds. The bold numbers indicate the algorithms with the best performance or smallest generalization gap.

(e.g., in the action/observation space and the reward function) of previous methods and therefore enables comparing different methods uniformly. See detailed description for the simulator and the environment settings in appendix.

Experiment Setting. Our simulator is based on the LOB data of 100 most liquid stocks in China A-share market. The data collected from April 2022 to June 2022 is used as the training set, and the data collected during July 2022 and August 2022 are used as the validation and testing set respectively. The task is to sell 0.5% of the total trading volume of the last trading day in a 30-minute period randomly selected from a trading day. The agent makes a decision (i.e., placing orders) at the start of each minute. We train an universal model for all the stocks. The evaluation metric is the trading cost defined as $(\bar{p}_{\text{TWAP}} - \bar{p})/\bar{p}_{\text{TWAP}}$, where $\bar{p} = A_a/V_a$ is the average execution price of the agent and A_a, V_a are the trading money and volume of the agent respectively, $\bar{p}_{\text{TWAP}} = \frac{1}{T}\sum_{t=1}^{T} p_t$ is the time-weighted average price in the given T time steps. Trading cost is measured in basis point (bp) which is 10^{-4}.

Baselines. We compare our algorithm with some rule-based and RL-based strategies for trade execution, where *TWAP* divides a large order into smaller orders of equal quantities and executing them at regular intervals throughout the entire period, *Momentum* buys relatively more quantity when the price rises, and vice versa.

The experiment results are shown in Table 2. We implement two families of algorithms based on DQN (which represents value-based RL methods) and PPO (which represents policy-based RL methods) respectively. Most of the previous RL-based trade execution algorithms are based on these two base RL algorithms. We implement these algorithms by

Predicted Statistics	Price	Volume
Avg future twap - current twap	4.5540	0.0971
Max future twap - current twap	1.5008	-0.2111
Min future twap - current twap	-1.0073	0.3853
Twap volatility	1.3095	-0.4248
Avg future sprd - current sprd	-0.1326	0.1083
Max future sprd - current sprd	1.8983	-0.1352
Min future sprd - current sprd	-1.3171	0.0292
Sprd volatility	6.3998	-0.8049

Table 3: The impact of predicted statistics on the agent's action in CASH (based on PPO).

following their designs in the model architecture, the observation, the action space, the reward function, etc. Moreover, we conduct experiments on the combinations of different designs and develop two well-designed RL-based trade execution algorithms (i.e., tuned DQN and tuned PPO in the table). Later, we implement our algorithms based on these two baselines.

First, we observe that tuned DQN/PPO outperforms other DQN/PPO-based baselines due to better designs. Specifically, we found that using only MOs for trade execution leads to high trading costs to cross the ask-bid spread. Moreover, the design of the reward function has a significant impact on the performance of the model. Second, we observe that our algorithms outperform tuned DQN/PPO due to aggregating the context. Notice that OPD [Fang *et al.*, 2021] uses a teacher policy that is based on the perfect information to guide the learning of the target policy, which is similar to our algorithms in extracting information from future contexts. However, OPD suffers from a larger generalization gap since the guidance of the teacher policy is informative but may not be generalizable. In contrast, the guidance in our algorithms (hand-crafted or generated statistics) is designed to be both informative and generalizable.

The advantage of using hand-crafted statistics to guide the learning of the context encoder in CASH is that the meaningful context representations can lead to an interpretable learned policy. To interpret the learned policy, we estimate how each dimension of the representation (i.e., the predicted statistics) affects the selected quoted price and volume based on a set of collected context representations. We list the slopes estimated using linear regression in Table 3. In this way, we can examine the learned policy. For example, we can see from the first row that the quoted price is 4.554 bp higher for every unit increase in the predicted price trend.

To present the policy of the agent under different trends, we plot an example of trade execution, as shown in Figure 3. Roughly speaking, the agent liquidates evenly across the horizon. Meanwhile, the agent can adaptively place the order according to the trend and the timeline. For example, the agent tends to place LOs at higher price levels in the early stage of the horizon and becomes more conservative in the latter stage. Moreover, when there is a rising trend, the agent is more inclined to quote at a higher price to catch the trend. We also show how the baseline PPO algorithm liquidates (cf. the purple lines). Compared with our algorithm, the baseline

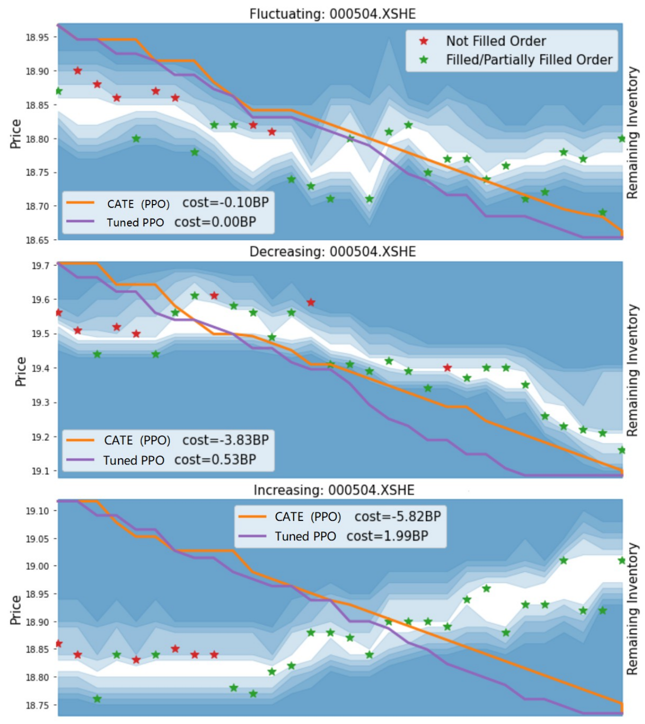

Figure 3: The policies under different trends learned by CATE (based on PPO) and the corresponding baseline (Tuned PPO). The background shaded areas indicate the 5-level ask/bid prices, and the stars indicate the orders placed by CATE. The lines indicate the remaining inventory of CATE and the baseline algorithm.

tends to liquidate more at some specific steps which may result from overfitting the training data. and the baseline tends to complete liquidation before the end of the given horizon which may lose trading opportunities.

6 Conclusion

To analyze the overfitting problem when applying RL to the trade execution task, we propose an Offline RL with Dynamic Context (ORDC) framework. In this framework, we derive the generalization bound for the ORDC and find that the generalization gap results from limited data and large context space. Motivated by the theoretical analysis, we propose to aggregate the context space to learn a generalizable agent. Accordingly, we design two algorithms: *CASH* that learns an interpretable agent using hand-crafted future statistics to guide context aggregation, and *CATE* that learns a compact context representation without resorting to domain knowledge. The experiments on both a simplified trade execution task and a well-designed high-fidelity simulated environment show that our algorithms can generate more generalizable agents. Moreover, combined with a better design on the model components (e.g., the reward function and the action space), our algorithms achieve significant performance improvement over the previous methods. In the future, we plan to apply the ORDC framework into other real-world RL applications that learn from offline context data.

Acknowledgements

Authors Yitong Duan and Jian Li are supported in part by the National Natural Science Foundation of China Grant 62161146004, Turing AI Institute of Nanjing and Xi'an Institute for Interdisciplinary Information Core Technology.

Contribution Statement

Chuheng Zhang and Yitong Duan have made equal and significant contributions to this work. Chuheng Zhang excelled in the development of the parts of theory and algorithmic, while Yitong Duan was instrumental in the parts of data processing and the simulator. Xiaoyu Chen conducted an exhaustive literature review. The original idea and concept of the paper were generously provided by Jian Li, Jianyu Chen, and Li Zhao. Their insights have been invaluable in shaping our research approach. We extend our gratitude for the collective effort that has culminated in this project.

References

[Almgren and Chriss, 1999] Robert Almgren and Neil Chriss. Value under liquidation. *Risk*, 12(12):61–63, 1999.

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

[Arjovsky *et al.*, 2019] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

[Bulthuis *et al.*, 2017] Brian Bulthuis, Julio Concha, Tim Leung, and Brian Ward. Optimal execution of limit and market orders with trade director, speed limiter, and fill uncertainty. *International Journal of Financial Engineering*, 4(1):175–200, 2017.

[Cobbe *et al.*, 2019] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[Cobbe *et al.*, 2020] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[Dabérius *et al.*, 2019] Kevin Dabérius, Elvin Granat, and Patrik Karlsson. Deep execution-value and policy based reinforcement learning for trading and beating market benchmarks. *Available at SSRN 3374766*, 2019.

[Dietterich *et al.*, 2018] Thomas Dietterich, George Trimponias, and Zhitang Chen. Discovering and removing exogenous state variables and rewards for reinforcement learning. In *International Conference on Machine Learning*, pages 1262–1270. PMLR, 2018.

[Du *et al.*, 2019] Simon Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudik, and John Langford. Provably efficient RL with rich observations via latent state decoding. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1665–1674. PMLR, 2019.

[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.

[Guéant *et al.*, 2012] Olivier Guéant, Charles-Albert Lehalle, and Joaquin Fernandez-Tapia. Optimal portfolio liquidation with limit orders. *SIAM Journal on Financial Mathematics*, 3(1):740–764, 2012.

[Kirk *et al.*, 2021] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

[Lin and Beling, 2020] Siyu Lin and Peter A Beling. A deep reinforcement learning framework for optimal trade execution. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 223–240. Springer, 2020.

[Lin and Beling, 2021] Siyu Lin and Peter A Beling. An end-to-end optimal trade execution framework based on proximal policy optimization. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 4548–4554, 2021.

[Mao *et al.*, 2017] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, 2017.

[Mao *et al.*, 2018] Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, and Mohammad Alizadeh. Variance reduction for reinforcement learning in input-driven environments. In *International Conference on Learning Representations*, 2018.

[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. PMLR, 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.

[Oroojlooyjadid *et al.*, 2022] Afshin Oroojlooyjadid, MohammadReza Nazari, Lawrence V Snyder, and Martin Takáč. A deep Q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, 24(1):285–304, 2022.

[Packer *et al.*, 2018] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

[Shahamiri, 2008] Masoud Shahamiri. Reinforcement learning in environments with independent delayed-sense dynamics. *Master Thesis, University of Alberta*, 2008.

[Shen *et al.*, 2020] Wei Shen, Xiaonan He, Chuheng Zhang, Qiang Ni, Wanchun Dou, and Yan Wang. Auxiliary-task based deep reinforcement learning for participant selection problem in mobile crowdsourcing. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1355–1364, 2020.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395. PMLR, 2014.

[Song *et al.*, 2019] Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. *arXiv preprint arXiv:1912.02975*, 2019.

[Wang *et al.*, 2020] Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. *Advances in Neural Information Processing Systems*, 33:7968–7978, 2020.

[Zhang *et al.*, 2018a] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[Zhang *et al.*, 2018b] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.