Experimenting with Reinforcement Learning in Market Making

An AFP Final Report

Team 5

Xinyu Li, Rongbing Liang, Yifei Tong, Luhua Yang, Zhiling Zhou

Faculty Advisor: Terrence Hendershott

March 9, 2022

Abstract

Market makers are the crucial players in providing liquidity to markets. They improve the market as well as earning profits by constantly quoting bid and ask prices in various market conditions. Researching the market making strategy has always been a hot topic among the academic and industry practitioners. In this project, we establish a simulated dealer market in equity basis and leverage it to test the behavior of different market participants including investors, naive market makers, adaptive market makers, and reinforcement-learning-based market makers. We showed that reinforcement-learning-based agent is able to learn from the market environment. It outperforms other agents by balancing skewness of bid/ask to earn spread P&L while being able to manage inventory simultaneously. Finally, to connect the whole empirical experiment to the real world, we proposed several improvements to the literature approach. The improvements consider the market impact of trades, the calibration of investor parameters to make it practical as well as asymmetric information in the markets. The proposed methods turn out to be a closer description of the real market.

Contents

1 Introduction **5**

1.1 Background 5

1.2 Motivation 6

1.3 Literature Review 7

1.3.1 Actions, Observations, and rewards 7

1.3.2 Various Market Agents and Their Behavior 8

1.4 Problem Statement 9

2 Proposed Approach **9**

2.1 Data Sources and Description 9

2.2 Outlining the Approach 10

2.2.1 Terminology and Definition 10

2.2.2 Simulating Environment 11

2.2.3 Basic Building Block 11

2.3 Reference Spread Curve 12

2.4 Adaptive Agent Implementation 15

2.5 Reinforcement Learning Agent Implementation 17

2.6 Connection to Real World: Market-Making Agents Enhancement 21

2.7 Connection to Real World: Investor Agents Enhancement 24

2.8 Connection to Real World: Comparison of Trading Statistics 26

2.9 Model Variations 26

3 Empirical Results **27**

3.1 Noise Investors Only 27

3.2 Addition of Momentum Investors and Mean-reversion Investors 29

3.3 Addition of Informed Investors 31

4 Conclusions and Future Work **33**

A Simulation Environment Parameters Settings **36**

B Graphs for Experiment: Addition of Informed Investors **36**

1 Introduction

1.1 Background

In this project, we apply reinforcement learning to the problem of market making as a study of whether an market maker trained by reinforcement learning can study the trading environment and strategies of other market participants well enough to outperform the others. For readers who might not be very familiar with reinforcement learning, we will briefly discuss the definition of the term, its application in the finance domain and its limitations.

Reinforcement learning, as a type of machine learning techniques, is quite similar to the concept of behavioral conditioning in psychology, where the subject of learning acquires knowledge in how to behave in a certain environment after a series of rewards to desirable behaviors and/or punishments to undesirable behaviors. The subject of learning, in this case, is an agent simulated by the computer, which is also the reason why reinforcement learning is almost always mentioned with another concept, agent-based modeling. According to Bonadeau (Bonabeau, 2002), agent-based modeling involves modeling of a system that is a collection of "autonomous decision-making" agents. These agents observe the system or the environment and make decision based on certain sets of rules. Agent-based modeling has been very popular in the modeling of financial markets as the interaction between agents simulates the real-life interaction between market participants.

Therefore, applying reinforcement learning is nothing but new to the field of financial research. A series of financial literature, including one that we will highlight in a later section, have discussed the application of reinforcement learning to market making, inventory risk management, high-frequency trading, and etc. In a lot of discussions, one major limitation of this agent-based approach stands out, that is given that the agent-based modeling approach relies on a simulated environment and a set of abstracted, rule-based agents, there is no guarantee that the results of an agent-based modeling approach can be translated to any concrete conclusions in the real-world. This limitation is therefore especially significant for a project that is supposed to generate insights in the real-world. Indeed, as you read on,

you will notice that we circle back to this issue repeatedly and discuss various ways that we can address it.

1.2 Motivation

As powerful as machine learning is, financial practitioners are often extremely cautious when applying machine learning techniques to solving real-world financial problem. Therefore, we believe that our motivation for applying reinforcement learning, a machine learning method, to the problem of market making should be deliberately explained.

Out of the many reasons why the financial industry shies away from machine learning in real world, the "black-box" nature or the lack of explanability is one of the most important ones. However, we believe that in our use case, explanability is less of an issue as the best actions learned by the agent will be based on the average reward of the actions through all rounds of simulation. Therefore, the connection between the data and the model is more visible to the human eyes.

Another reason we decide to apply reinforcement learning to this problem is that in the context of market making, environment, opponents, and rewards can all be clearly defined, abstracted, and observed, which all make reinforcement learning sound like an ideal candidate to solving this problem.

Moreover, perhaps the most important reason we feel motivated to conduct this study is that we believe there are approaches we can adopt or extra model specifications we can add to our agent-based model that can break the barrier between the simulated agent-based environment and the real world or at least provide reasoning that such a break is feasible. We have thought about ways that we can achieve this from two fronts. The first one from the model front by adding rules and specifications to our simulated agents so that they act more realistically. The second one from the result front by comparing the simulated results with real market data while adjusting model parameters accordingly such that the simulated and real market dynamics look similar. In later sections, we will discuss specifically what types of approaches we think about and which ones we end up adopting given the limitation

of our dataset.

1.3 Literature Review

Our idea of training a market maker agent in the dealers' market was originally inspired by a paper authored by several researchers from JPMorgan AI Research (Ganesh et al., 2019). The paper discussed a series of experiments that aimed to compare the performance of a market maker agent trained using reinforcement learning in the simulated environment relative to other market maker agents with much simpler mechanisms of price quoting and hedging. To the best of our understanding, the paper focuses mostly on purely simulated environments and it does not seem to make much effort in discussing the extent to which the simulated environments resemble the reality.

While we believe that the lack of established connection between the experiments and the real world needs to be carefully addressed in our AFP project, we do find that the paper lays some solid groundwork for building a simulated environment. Specifically, the Garesh paper provides clear definition of a market maker's actions, observations, and rewards in this multi-agent system, as well as outlining some easy-to-implement but also potentially realistic behavior for various types of market makers.

1.3.1 Actions, Observations, and rewards

The main actions we need to consider for a market maker in the simulated dealers' market are the bid and ask prices the agent quotes for different sizes of request for quotes (RFQ). These can be defined in many different ways. The paper by JPMorgan AI Research (Ganesh et al., 2019) defines the two prices in terms of the percentage differences (referred to as buy/sell spreads) between the quoted buy/sell prices to a mid-price which is simulated with a Geometric Brownian Motion. The paper also further abstracts the buy and sell spreads with the difference between the agent's buy/sell spreads vs. the best available buy/sell spreads in a hypothetical exchange market.

Another important element in a market maker's action space is the hedging ratio, namely

the percentage of the current inventory an agent wishes to hedge against.

The rewards for a market maker are quantified with a single P&L-like value which is the sum of three different parts: i) Spread P&L: spread profit earned by the market maker, ii) Inventory P&L: the profit or loss generated by the agent inventory's exposure to asset price movements, iii) Hedging cost: the cost paid by the agent to hedge against inventory risk.

In a multi-agent reinforcement learning framework, according to the paper, a reinforcement-learning-based agent observes actions and associated rewards from the past time steps and chooses the best actions according to past average rewards for each action.

1.3.2 Various Market Agents and Their Behavior

Besides the RL-agent, the Garesh paper also discusses actions/behavior for several other types market agents.

- Random Agent: select both buy/sell spreads and hedging ratios from uniform distributions (the choices of parameter values discussed in more detail in later sections)
- Persistent Agent: maintain a constant buy/sell spreads (the choices of parameter values discussed in more detail in later sections)
- Adaptive Agent: uses a rule-based optimization method to select best buy/sell spreads to meet its targeted market share and to internalize the inventory risk then chooses a hedge ratio that optimizes a risk-aversion adjusted version of total P&L

Investor agents are the liquidity demanders or the counterparties to the market making agent. The paper assumes a simple model of 20 classical "noise trader", each of which buys or sells 1 unit of the security with 50% chance. This liquidity demand model is definitely far from the underlying model of a real market. To address this issue, we implement 3 additional types of investor agents which will be described in a later section.

1.4 Problem Statement

As an attempt to extend the research done by the above-mentioned paper and address some of the issues that we believe deserve more discussions, our project set out to study three key questions:

1. How do we measure the similarity between a simulated market and the real-world market?
2. Are market making agents trained by reinforcement learning able to learn the strategies of other market participants and outperform them?
3. Starting from the simplest simulated market environment and progressively moving towards more complicated and realistic ones, how will the increase in complexity of the environment affect the learning ability of RL-based agents?

2 Proposed Approach

2.1 Data Sources and Description

Before discussing details of our approach to solving the problems outlined in section 1.3, we would like to provide a brief description to the dataset we have collected. We have decided to study the dealers' market in the context of stock market space mostly because we want the assets we study to be traded frequently enough for the RL-based agent to study while at the same time we also wish to include some effects of asymmetric information in the market. Additionally, we want to be able to access a dataset with LOB data in order to fit a reference LOB curve that will help simplify the simulation of price quoting and hedging of various agents. To this end, we have chosen a dataset of stock trading book and limit order book collected from OneTick on MSFT. However, due to the limitation of this data source, we were only able to collect data for 14 non-consecutive days.

The raw data set consists of 10 level bid-ask quote prices and volume for each selected stock over the period: 08/24/2017-09/06/2017, which has 14 trading days and approximately 7.5

million data points in total. To approach the normal market condition, we filter out data points beyond 9:30 AM to 3:50 PM and resample the date at a 1 second frequency.

However, it is important for us to note that the dataset we collected is not the same as the one used by the paper we referenced. The most notable difference is that our collected dataset is from the stock exchanges only and have no entries from the dealers' market. This discrepancy creates troubles for us when we try to compare our simulated environment with the real market.

2.2 Outlining the Approach

In this section, we will provide an outline of our approach. As we study the behavior of an RL-based agent in a simulated environment much like that in the paper we discussed, we adopt much of the paper's methodology in creating a simulated environment. However, to better fit the approach our problem statement and to concede to the limitations of our dataset, we make several adjustments to the approach.

2.2.1 Terminology and Definition

Quick reference for symbols and terminologies used later in the report:

Buy/Sell Spread Percentage Diff vs. the Reference Spread	$\epsilon^b(v)$ and $\epsilon^s(v)$
Hedge ratio	x
Trade size	v
Inventory	z
Dollar spread	s
Best price dollar spread	s_0
Reference dollar spread from LOB	$S_{ref}(v)$
Spread P&L	$v \times s$
Hedging Cost	$x \times z \times s$

2.2.2 Simulating Environment

To create a simulated environment where a RL-based agent can be trained, we adopt the behavior of the random agents, persistent agents, adaptive agents and investor agents from the paper Ganesh et al., 2019.

We also adopt the paper's assumption that the mid price follows a geometric brownian motion. In order to make the mid price movement more realistic, we decide to also incorporate the effect of trading that happens within each time step to the movement of the mid prices. The way we do this is at the start of each time step calculate the theoretical value of next mid price assuming it strictly follows the geometric brownian motion. After a time step of simulation, calculate the adjusted mid price as a weighted average between the last trade price of the simulation and the theoretical next mid price.

2.2.3 Basic Building Block

Four main building blocks are together responsible for building up the entire simulation system. They are market-making agents, investor agents, epochs, and simulations. don We design market-making agent as an interface with subclass such as random agents and persistent agents each implemented with their own version of price quoting and inventory hedging while sharing common behavior like appropriately updating P&L after winning a trade.

Investor agents are designed in a similar way with a base-class sitting on top and subclass such as noise investors and momentum investors each implemented with their own function to submit market orders based on market conditions.

Epochs represent a time interval or a round in our market simulation. In each epoch, the market mid-price and other market condition variables are recalculated. Based on the change in market conditions, each investor agents submit their buy and sell orders while each market-making agent submit quotes to win orders in the market. That is to say that the agents will not change their quotes until they reach the next epoch. For RL-based agent

specifically, the learning/training process also happens once at the beginning of each epoch. We also decide that each epoch will be 15 minutes long. We arrive at this length of time after thorough discussion within the group and with Professor Hendershott. We believe that an 15-minute interval achieves a balance between frequency for agents to update quotes and frequency of the RL training in order for the actions of the RL-based agent to converge.

Simulation is the main class that represents one set of simulation. It stores macro information about the simulation environment, which includes the drift and volatility of the mid-price process, the list of market participants, the length of an epoch, and etc. Each simulation will manage its own list of epochs, with each epoch building on last one in the list. The simulation class is therefore responsible for passing all the information from epoch to epoch and from epoch to agents.

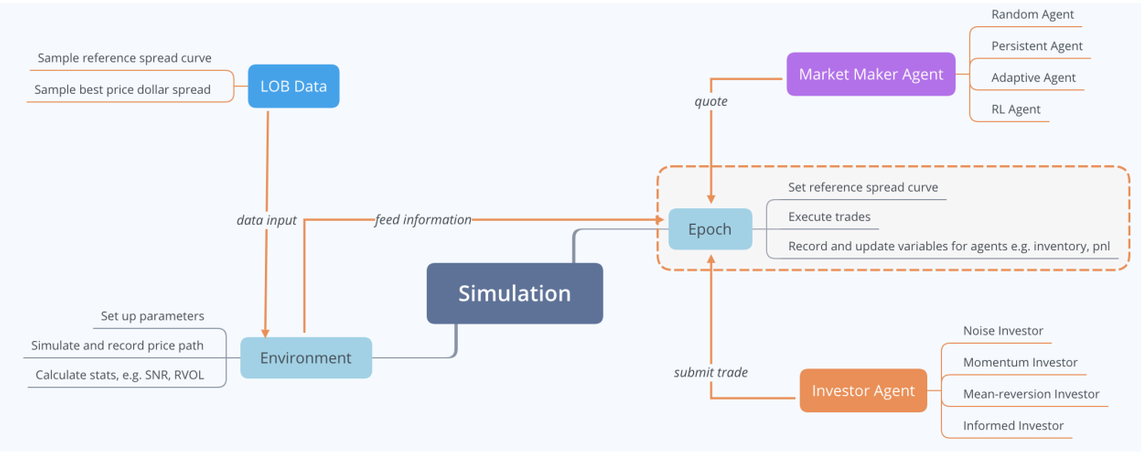

Figure 1: Building blocks of simulation environment

2.3 Reference Spread Curve

In the Ganesh paper Ganesh et al., 2019, the reference spread curve represents a LOB structure and is assumed to exist as a parallel to the dealer's market. The significance of the reference spread curve originates from each agent's need to quote with respect to it. As mentioned previously, in our simulation, each agent will not submit a buy/sell price or a buy/sell spread. Instead, the agent will submit a percentage difference with respect to the current reference spread in the exchange. Therefore, fitting a reference spread curve from the LOB data we received becomes the first important connection between our simulation and the real market.

The reference market spread is defined as $s_0 = a_0 - b_0 = 2S_{ref}(0)$, and the reference market mid price is given by $P_t = a_0 + b_0/2$. We assume the market reference price curve $S_{ref}(v)$ to be known by all participants and specifies the cost of trading a size v on the reference exchange (relatively to the mid price). And we also assume that $S_{ref}(v)$ is the same on the bid and ask side.

The reference market spread s_0 are sampled from the real market data at each time step. Basically, we group the sample distribution of spread by 15-minute time interval. For each epoch, we take one sample spread from the corresponding time interval as the reference market spread. In this way, the dynamic of the spread will approximate the real market spread dynamic, which is more likely to have greater value at the opening.

As for the best level volume v(0). We fit it by the gamma distribution for each time interval. And we also fit the shape of LOB by a 2nd order polynomial function to get the calibrated sample shape functions for each time interval. Specifically, for each time interval and each date, we minimize the sum of square error of the following function with the constraint that shape(0)=1:

$$shape(n) = b_0 + b_1 n + b_2 n^2$$

In this case, we will get 14 sample shape functions for each time interval. Thus, at each time step, we sample one shape function and one best volume by the corresponding gamma distribution. Then we model 10 levels of bid-ask volume given $s_0/2 + n$ tick (one tick=0.01) by the following function:

$$v(n) = \lambda * v(0) * shape(n)$$

$$shape(0) = 1, v(0) \sim \Gamma(\alpha, \beta)$$

The scaler lambda is a constant currently calibrated before the experiment such that investor trades process are able to hit the full range of LOB levels.

For illustration purposes, we display one day samples of the LOB shape function in Figure 2, which are calibrated by the LOB data of the stock MSFT. It describes the relative volume scale of each tick level to the best level volume for each time interval. Interestingly, we find

the shape around the opening period will be more upward-sloping, which means the top-levels of the LOB will have more liquidity, suggesting a greater impact of the information asymmetry.

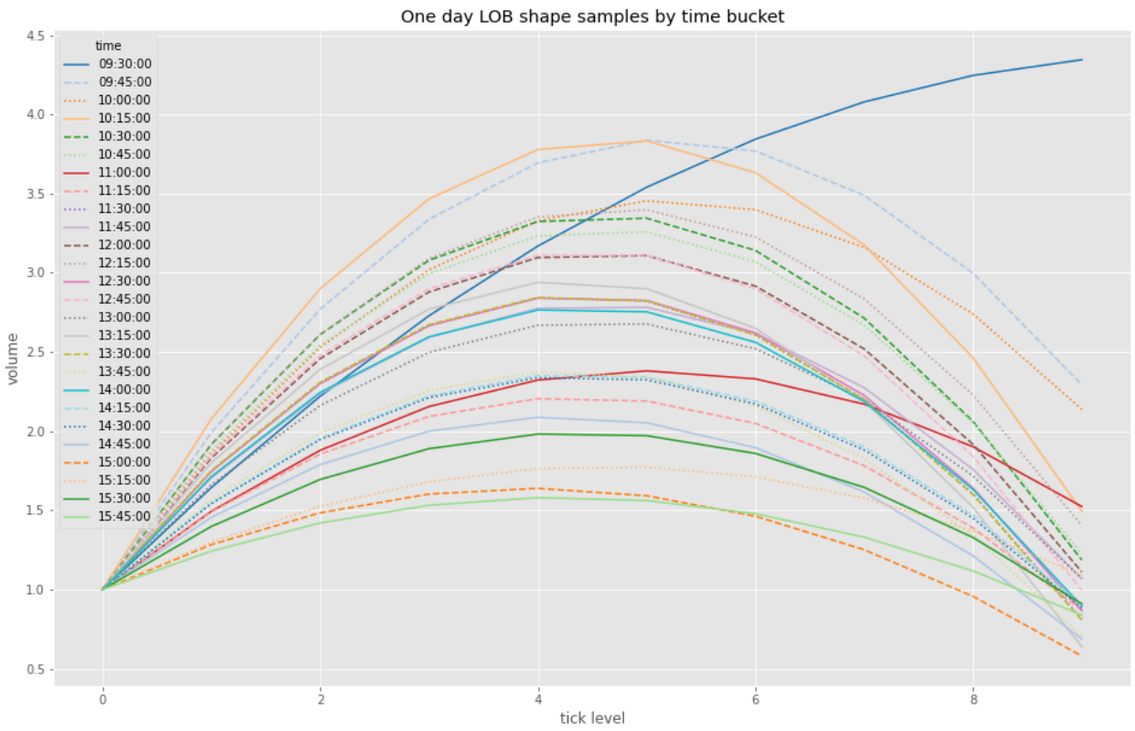

Figure 2: MSFT LOB Shape by Time Bucket

Moreover, we also take out four typical time buckets, and display the distribution of the best level volume and spread respectively in Figure 3 and Table 1. The result shows that the opening period is likely to have less volume and larger spread, while the closing period has more liquidity available.

bucket_label	count	mean	std	min	25%	50%	75%	max
09:30:00	12600	0.013	0.007	0.01	0.01	0.01	0.01	0.12
11:30:00	12600	0.010	0.001	0.01	0.01	0.01	0.01	0.02
13:30:00	12600	0.010	0.001	0.01	0.01	0.01	0.01	0.02
15:45:00	11773	0.010	0.001	0.01	0.01	0.01	0.01	0.02

Table 1: MSFT Reference Market Spread Statistics

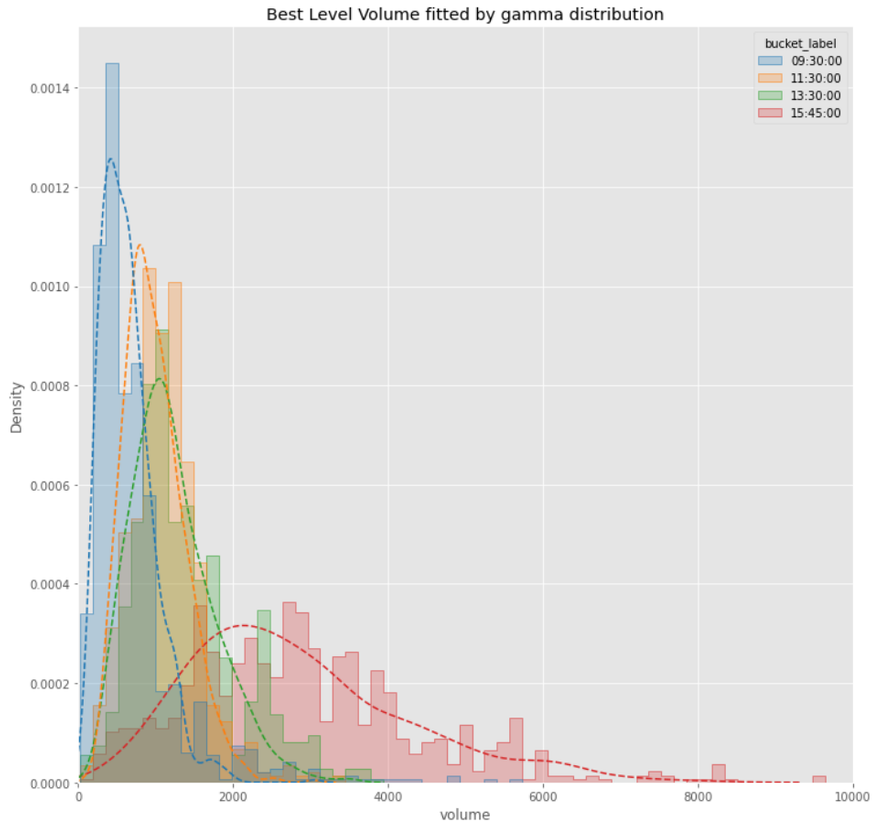

Figure 3: MSFT Best Volume Distribution

2.4 Adaptive Agent Implementation

We adopt the similar adaptive pricing and hedging policy as in Ganesh et al., 2019.

At each time step, the agent chooses its bid/ask spread ϵ^b, ϵ^s to meet its targeted market share η_{ms} within a tolerance $\delta_{tol} = 5\%$ and internalize by skewing its prices using the following 2 steps.

1. Solve

$$\epsilon_* = \max\{\epsilon : |cost_1(\epsilon) - min_x cost_1(x)| \leq \delta_{tol}\}; cost_1(\epsilon) := |\eta_{ms} - \frac{E[v_\epsilon]}{V_{market}}|$$

   Where V_{market} is observed from the market volume of the last time step, and $E[v_\epsilon]$ is estimated by Gaussian KDE using data in previous time steps.

2. The agent then skews only one side to attract a flow that offsets its current inventory z. It choose the spread by minimizing

$$cost_2(\epsilon) := -S_{ref}(0)E[s_\epsilon] + \gamma\sqrt{S_{ref}(0)^2 var[s_\epsilon] + \hat{\sigma}^2 E[(z + v_\epsilon)^2]}$$

   Call the result of the optimization ϵ^{**}.

Where $E[s_\epsilon]$, $var[s_\epsilon]$ and $E[(z+v_\epsilon)^2]$ are estimated by Gaussian KDE using data in previous time steps.

If $z > 0$, $\epsilon^b = \epsilon^*$, $\epsilon^s = \epsilon^{**}$; If $z < 0$, $\epsilon^s = \epsilon^*$, $\epsilon^b = \epsilon^{**}$; Otherwise, $\epsilon^b = \epsilon^s = \epsilon^*$.

At each time step, the agent chooses the hedging ratio x by minimizing

$$cost_{hedge}(x) := |xz|S_{ref}(xz) + \gamma\sqrt{\hat{\sigma}^2 E[(z(1-x)+v_\epsilon)^2]}$$

Where $E[(z(1-x)+v_\epsilon)^2]$ are estimated by Gaussian KDE using data in previous time steps. And risk aversion coefficient $\gamma = 2$ according to Ganesh et al., 2019.

A further discussion on why we use the Gaussian KDE is necessary. For adaptive agents, we need to come up with an empirical estimation of the mean and variance of the incoming net flow v_ϵ and the normalized spread P&L s_ϵ. These two estimations are a function the bid/ask spread pair ϵ_b, ϵ_s. Therefore the best approach is to estimate the probability density function with respect to the variable we are interested as well as the bid/ask spread. Based on that, every time we have a pre-specified bid/ask value, we could input these and do integral on the probability density function, the integral variable is the variable we are interested.

In statistics, the estimation of probability density function of a random variable can be classified into parametric approach and non-parametric approach. The parametric method requires a prior distribution of parameter as well as an assumption of the form of probability density function. If all these are reliable, the estimation should be more precise compared with non-parametric methods. However, the mechanism between bid/ask spread and the incoming flow and normalized spread is too complicated to express in an analytical way. Therefore we turn to the non-parametric approach.

In terms of non-parametric approach, the kernel density estimation is one of the most powerful and popular approach. The idea is for every independent and identically distributed sample x_i, we could know its contribution to the probability density across the sample space. Apparently the impact will be the most significant around the collected sample and decay

fast if moving away. The relationship can be formed as

$$\widehat{p(x)} = 1/n \sum_{i=1}^{n} K(x - x_i)$$

where K is the kernel we need to assign, it only depends on the distance between the estimation point and the samples, so it's a non-parametric approach. In our experiment, we leveraged the Gaussian kernel. Gaussian kernel is versatile and has been widely used. As long as we collect enough data, the probability density function should converge to optimal.

2.5 Reinforcement Learning Agent Implementation

1. Q Learning

   We first tried the traditional Q learning method with tabular table. We have a 2D state space, including the inventory, the market volatility. For inventory, we divide our inventory into discrete buckets, each bucket represents different risk level of our appetite. For example, in our design, we believe that 25 is a risk limit for our model, therefore our inventory space will aggregate all numbers larger than 25 into one single category, and ideally the model should be able to learn that we need to offload the inventory when we are about to reach the limit. For volatility, we compare the current level with historical level and decide whether it is higher or lower than the average level. By having this basic matrice in mind, we calibrated the market volatility based on the market spread, and classified it into five categories based on its relative intensity, as shown in table 3. Instead of putting the real value for market volatility, we discovered that making it into categories helps with the model convergence.

Inventory
Smaller than -25
-25 to -15
-15 to -5
-5 to 5
5 to 15
15 to 25
Greater than 25

Table 2: Inventory States

Volatility
Extra high
High
Medium
Low
Extra low

Table 3: Volatility States

For action space, we have 2 dimensions as well: skew and spread. Spread 0 means we are quoting at the mid, spread 1 means we are quoting at the maximum predetermined spread. The spread action can help control our aggressiveness and respond to market volatility. Another action is skew, this action is primary related to our inventory and risk appetite. If we are very long in a product, we would like to lower down both our bid and ask to reduce the inventory, and we call it "skew down". Similarly. when we are short in a product, we would like to skew up. The amount of skew we want to have depending on our size of inventory and risk appetite.

Spread
0
0.25
0.5
0.75
1

Table 4: Spread Actions

Skew
+0.25 — Skew up
0 — Flat
-0.25 — Skew down

Table 5: Skew Actions

After we have defined our states and actions space. We train our Reinforcement Learning model using Temporal Difference and self defined reward function.

2. Reward Function

The reward function is one of the most important inputs to an RL training algorithm.

Our reward function consisting of 3 components, each of them responsible for difference behaviors we want to achieve on the RL market maker. Ideally, the RL market maker should be able to find a good balance between those constraints. In our case, the biggest trade-off is between the Spread P&L and Risk Aversion. The more trades we want to aggressively capture, the more likely we are going to have a greater exposure. The less exposure I want to have and be more conservative, the less spread I am able to capture and therefore a smaller market share we have.

i. Spread P&L

Spread P&L is the profit we capture from actively quoting at two sides of the theoretical price and providing liquidity to the market. Here the spread P&L is a notional P&L we make based on the current market mid price. We will also need to hedge our position in the end of the day to know our exactly P&L.

$$Spread\ P\&L = bid\ spread \times bids\ captured + ask\ spread \times asks\ captured$$

ii. Inventory P&L

The inventory P&L is the P&L due to the underlying price change. When we are making the market, we are also exposing us to the delta risk. Often times, we are not able to fully hedge our position in a smooth fashion so we always accumulate some amount of inventory P&L in the end of the day.

$$Inventory\ P\&L = inventory_t \times return_t$$

iii. Risk Aversion

This is a self defined term that we use to regulate the Reinforcement Learning model. This term tells the RL model to reduce the inventory exposure regardless our inventory P&L. Sometimes the model is able to make a good bet on the direction of the market and speculate. However, as a market maker, we do not want aggressive exposure to the delta. We would prefer a less risky way of making money. Therefore we add this term in the reward function to reduce inventory exposure. In the experiment, we can flexibly change the value of λ to make a more risk averse or risk appetite RL market maker.

$$Risk\ Aversion = -1/2\lambda \times inventory_t \times volatility_t$$

3. Training Algorithm: Temporal Difference

We use the traditional Temporal Difference method to train the reinforcement learning model. The training algorithm will output a table called "Q table", the larger the Q value, the better the action is for certain states. The trained RL model will reference the Q table and choose the highest Q value action for every time step. Here is an illustration of our Q table, we selected several typical states and their corresponding best actions.

States		Actions		
Inventory	Volatility	Spread	Skew	Q Value
-25 to -15	High	Wide, 1	Skew up, 0.25	1
-5 to 5	Medium	Medium, 0.5	Flat, 0	1
Greater than 25	Low	Tight, 0.25	Skew down, -0.25	1

Table 6: Snapshot of Q table

This is the pseudo code for our Temporal Difference training algorithm:

Q-learning: An off-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 4: Q learning Temporal Difference Algorithm

4. Value Function Approximator

The last milestone for our reinforcement learning model is to design a value function approximator for our states and actions space. The idea is as follows: We used to have a table that store the discrete states and actions and our Q values. But in real life, the states and actions are rather continuous. How are we going to input those continuous state and action space into Q learning? We cannot use the previous approach because it will introduce a much larger space and reduce the convergence speed exponentially.

Instead, we use a regressor, to "model" the real relationship between the states and actions. Let us assume that there is some real value function that will map the state space to the action space. The goal of the SGD regressor here is to fit the value function. Additionally, we made an aggressive assumption that the relationship is linear and we use a linear regressor to reduce the training time. As a result, we are able to find the actions for all different continuous states and make the model more realistic. However, the idea is still the same with tabular method, that is to find the best Q value action under each state.

2.6 Connection to Real World: Market-Making Agents Enhancement

Like we discussed in section 1.2 and section 1.3, we find that establishing a connection between the simulated environment and the real market should play a significant role in our project and our approach to this goal should be also carefully deliberated.

The first set of approach we think about to establish such a connection is by enhancing the model with more real-life features or abstractions. Specifically, we wish to add specifications to our simulated agents such that they behave more like they would in the real world. We address both market-making agents and investor agents with this approach.

Among the different types of market-making agents, the adaptive agents and the RL-based agents can both learn and adapt to the environment. We view them as agents with real agency. On the other hand, persistent agents and random agents are more integrated to the environment. Therefore, as an attempt to create a realistic environment, we put our focus on persistent agents and random agents.

For persistent agents, the parameters we need to decide are the constant ϵ^b and ϵ^s (see section 2.2.1). We decide to hold these two values close to 0 as a floor for spreads quoted by agent in order to win a trade.

We believe that random agents, on the other hand, are the source of noise in the price

process. Adding a random agent to the environment and allowing the intra-step trading to affect the price process, in our opinion, can make our price process more realistic than simply assuming a geometric brownian motion by taking into account trading noise. We outline the detailed steps through which we apply this enhancement below.

First of all, we wish to make sure that our theoretical geometric brownian motion has the correct parameters. To do this, we take the market price variable from our collected stock dataset and compute 1) average daily log return over the 14 days for which we have the data for as a daily drift, and 2) average daily realized volatility with a 15-minute sampling interval as daily volatility. We then input these two values as the drift and the volatility to our simulation environment as the parameter for our geometric brownian motion.

After we fix the geometric brownian motion as our theoretical price process, we then can start running a simulation with 1 random agent, 1 persistent agent, and 1 adaptive agent. In this simulation, we identify three key factors that can affect how the noise of intra-epoch trading gets incorporated into the price process. These processes are ϵ_{min}, ϵ_{max} (the upper and lower bound for the uniform distribution of the random agent), and a price impact coefficient (PIC). To explain what the PIC is, we need to go back to how we implement the mid-price updating process in our simulation environment. At the start of each epoch, we have a mid-price from the previous epoch or an initial mid-price if it is the start of a simulation. We compute a theoretical mid-price for the next epoch using the formula of geometric brownian motion. Then after all of the trading activities, the size-weighted average of last buy or sell price of the epoch will be recorded as the post-trading mid-price for the epoch. Finally, before we start simulating the next epoch, we compute the mid-price for the next epoch as

$$\text{Mid-price}_t = (1 - \text{PIC}) * \text{Theoretical Mid-price}_t + \text{PIC} * \text{Post-trading Mid-price}_{t-1}$$

Essentially, the PIC serves as the weight for post-trading mid-price in a weighted average of the post-trading mid-price and the theoretical mid-price.

Now that we identify the variables affecting the noise in the price process, how do we match

the noise level of the real price process and the simulated price process? Here we introduce a metric we called signal-to-noise ratio (SNR) and the ratio for one day is computed using the formula below. Assuming we divide a one-day price time series into n intervals numbered 1 to n,

$$\text{One-day SNR} = |\text{price}_n - \text{price}_1| / \sum_{i=1}^{n-1} |\text{price}_{i+1} - \text{price}_i|$$

Using this formula, for example, we computed the average 1-day SNR for MSFT to be about 0.14. To create a simulated environment with a mid-price process close to the real MSFT process, we use grid-search over ϵ_{min}, ϵ_{max} for random agent (ϵ^b, ϵ^a for persistent agent) to generate a simulated environment with an average (with over 10 samples) 1-day SNR that is close to 0.14. In the case that all the investor agents are consisted with noise investors, during the process of grid search, we notice that fixing either ϵ_{min} or ϵ_{max} while changing the value for the other variable, the relationship between SNR and the other variable seems to be monotonic (see figure 5 and 6). The set of variable values that output the closest SNR will be the input to the random agent, persistent agent and the simulated environment. The parameters are summarized in appendix A.

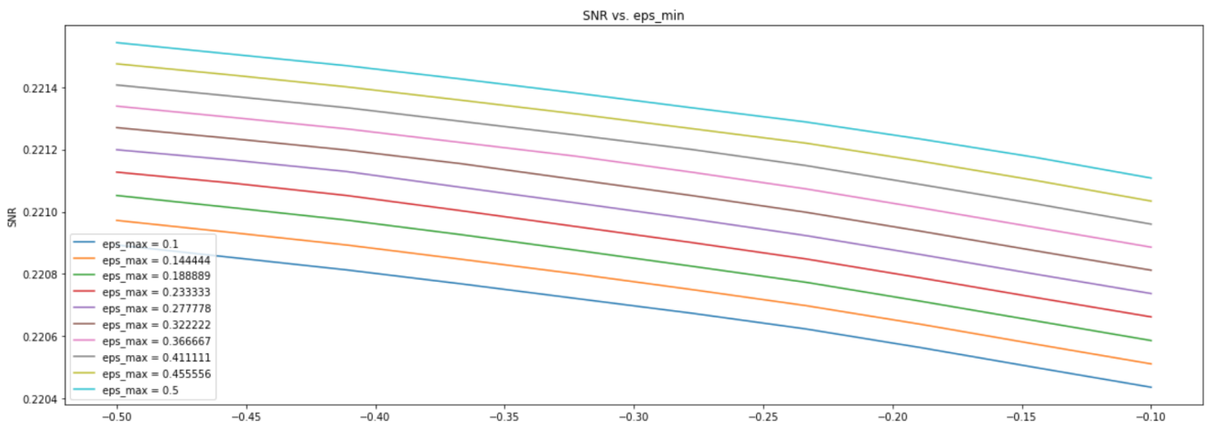

Figure 5: SNR vs. ϵ_{min}

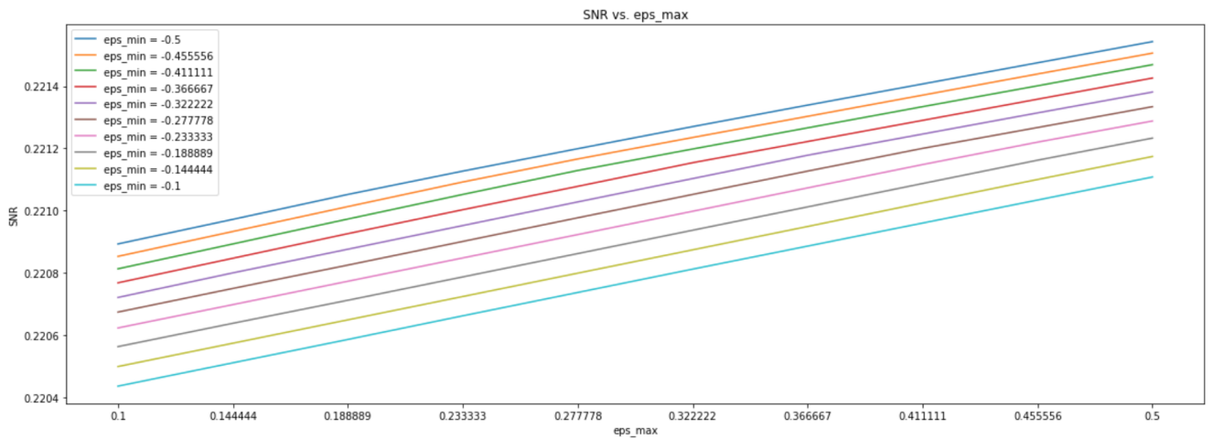

Figure 6: SNR vs. ϵ_{max}

2.7 Connection to Real World: Investor Agents Enhancement

In the reality, we do not believe that the market is full of random "noise" investors, which has 50% of probability to buy or sell, like it is specified in the Ganesh paper Ganesh et al., 2019. Sure an aggregation of retail investors in the real world might look like a set of random investors but in a market like the stock market where information is so easily spread, even the retail investors can have investment rationale or can be informed. In this section, we introduce three additional types of investor agents to our model.

The first two types are similar but different. They are momentum investors and mean-reversion investors. They are similar in that they both base their trading action on the information of the last epoch but they are different in the direction of their actions.

Here we assume that the return distribution for our security is normal and we estimate the mean and the standard deviation of this distribution using a 10-epoch window. The momentum investors submit orders according to the following rules:

- If $mean - 1 * std \leq return \leq mean - 1 * std$, no order is submitted
- If $mean + 1 * std \leq return \leq mean + 2 * std$, buy with 70% chance
- If $mean + 2 * std \leq return \leq mean + 3 * std$, buy with 90% chance
- If $return \geq mean + 3 * std$, buy with 100% chance

- If $mean - 2 * std \leq return \leq mean - 1 * std$, sell with 70% chance
- If $mean - 3 * std \leq return \leq mean - 2 * std$, sell with 90% chance
- If $return \leq mean - 3 * std$, sell with 100% chance

Similarly, the mean-reversion investors submit orders according to the following rules:

- If $mean - 1 * std \leq return \leq mean - 1 * std$, no order is submitted
- If $mean + 1 * std \leq return \leq mean + 2 * std$, sell with 70% chance
- If $mean + 2 * std \leq return \leq mean + 3 * std$, sell with 90% chance
- If $return \geq mean + 3 * std$, sell with 100% chance
- If $mean - 2 * std \leq return \leq mean - 1 * std$, buy with 70% chance
- If $mean - 3 * std \leq return \leq mean - 2 * std$, buy with 90% chance
- If $return \leq mean - 3 * std$, buy with 100% chance

The chances at each boundary roughly follows the pdf of a normal distribution.

To study how our RL-based agent performs in an environment with asymmetric information, we introduce one final type of investor agent called informed investors. These investors have a simple behavior. At the start of each epoch, they will know in advance what the theoretical mid-price given by the geometric Brownian motion will be for the next epoch. They do not, however, have the ability to predict how the current epoch's trading activities will affect the mid-price. If they see an upward price movement from this epoch to the next, they submit a buy order and vice versa.

And for all the investor agents, their trade size will follow

$$v = a \times Poisson(-k \times s_0), a > 0, k > 0$$

where a is the scaling parameter for size and k is the scaling parameter for distribution. We choose Poisson distribution because of the fact that order flows are usually assumed to be Poisson process. This formula is based on the intuition that, the smaller the best price dollar spread, the larger the trade size. We set $a = 10$ and $k = 1$ in all the experiments.

2.8 Connection to Real World: Comparison of Trading Statistics

We mentioned in section 1.2 that we will discuss the connection between the simulated environment in two fronts. While the previous two sections focus on the model front, in this section, we discuss ways we think about to establish the connection between the simulation and the real market by comparing key trading statistics from the results.

One way we can compare the two is by computing trading frequencies of an epoch. Given a simulation with enough epochs, we can compute the frequencies of trades for each epoch. This will generate a distribution of trading frequencies within an epoch. A similar distribution can be constructed using real market data. By comparing main moments of these two distributions, we can get an idea of how similar two trading environments are in terms how frequently a trade gets done.

More sophisticated comparisons we can do include how often random agents or persistent agents win a trade, market shares of adaptive agents, and etc. However, these comparisons do require more detailed data including the origin of quotes and identity of agents from the real market.

While we listed several ways we can compare the end results of our simulation and the real market data, we do want to put on notice that our collected dataset do not include any trading data from the equity dealer's market, which makes the comparison impossible. We do believe we can finish the comparison of trading frequencies if we had access to a dealer's market dataset.

2.9 Model Variations

In later sections, we will show and discuss the results of our simulations. The results will be organized in terms of growing complexity of model.

The order of models we consider is:

1. Simple model with 1 random agent, 1 persistent agent, 1 adaptive agent, and noise

investors only. The investor agents are consisted of 20 noise investors.

2. Addition of momentum investors and mean-reversion investors. The investor agents are consisted of 10 noise investors, 5 momentum investors and 5 mean-reversion investors.

3. Addition of informed investors. The investor agents are consisted of 8 noise investors, 4 momentum investors and 4 mean-reversion investors and 4 informed investors.

We will also run versions of 1-vs-1 environments where we explore the robustness of RL-agent training and study whether a RL-agent trained in a more complex environment will be able to perform well in a simpler environment.

3 Empirical Results

3.1 Noise Investors Only

In this first model, we experiment with noise investors only. The key results of the simulation are shown below in table 6.

	Random	Persistent	Adaptive	RL
hedge cost	-0.0032	0.0000	-0.0015	-0.0006
spread P&L	0.4177	0.2107	0.0148	0.0086
inventory P&L	11.1268	-11.4223	0.1237	0.0461
total P&L	11.5413	-11.2116	0.1369	0.0542
reward	-34.3636	-35.2065	-0.7999	-0.2130

Table 7: All 4 agents performance per time-step

By the first look, it looks like the RL-based agent is doing the worst. By the judge of almost all P&L categories, the RL-based agent is the last. If we look at the numbers in more details, we can see that the main reason why the RL agent is not outperforming most of other agents in this scenario is because the RL agent does not try to win too many trades. During the training process, we have specified that the main objective function for the RL-based agent is given by a metric named reward, which is given by the formula below:

$$\text{Reward} = \text{Spread P\&L} - 0.5|\text{Inventory P\&L}|$$

By optimizing the agent over this objective, we are essentially training the agent to earn as much spread P&L as possible while maintaining a low inventory. With our optimization in an environment full of noise investors, the RL-based agent decides that the best action is to focus on inventory management by controlling the number of trades it tries to win. In this comparison, the RL agent is actually outperforming the other competitors as the table shows that the RL agent has the highest reward. If we look at the random agent and the persistent agent, both agent dominate in terms of spread P&L but it comes at a cost of having to maintain an extremely large inventory. The risk taken by these two agents are too high compared to the scale of spread P&L they are earning which is also evident in the comparison of the scale of inventory P&L and the scale of spread P&L for these two agents.

	Random	RL
hedge cost	-0.0014	-0.0003
spread P&L	0.9275	0.1193
inventory P&L	0.0581	-0.0030
total P&L	0.9842	0.1160
reward	-0.4963	0.0398

Table 8: RL agent vs Random agent performance per time-step

	Persistent	RL
hedge cost	-0.0003	-0.0008
spread P&L	1.0274	0.0244
inventory P&L	-0.1166	0.1296
total P&L	0.9105	0.1532
reward	-0.3807	-0.5394

Table 9: RL agent vs Persistent agent performance per time-step

We then put our trained RL agent to test in a series of 1-vs-1 situation against every other type of agents. In these scenarios, since there are only one other liquidity provider, the RL

	Adaptive	RL
hedge cost	-0.0033	-0.0003
spread P&L	0.6941	0.3244
inventory P&L	0.1089	-0.0133
total P&L	0.7997	0.3107
reward	-0.6059	0.1387

Table 10: RL agent vs Adaptive agent performance per time-step

agent is forced by the environment to take up more market shares. This is reflected in the higher spread PL values for the RL agent in each of the cases. The extent to which the RL agent expand its market share also depends on the only other competitor's aggressiveness in the market. In the case of RL agent against the adaptive agent, the RL agent is able to earn spread PL almost 40 times of that it earned in the case against all types of agents. The RL agent is also able to maintain a low inventory at the same time.

The only exception is with the persistent agent where the RL agent trained in the more complex environment failed to recognize the persistent agent is giving out constant quotes and it only needs to beat that quote by a small margin to win the trade. This results in a very low spread PL for the RL agent.

3.2 Addition of Momentum Investors and Mean-reversion Investors

By replacing some of the noise investors with both the momentum and mean-reversion investors to the model, we observe that the magnitude of P&L for all agents go down. This could be caused by the reducing number of market orders submitted by investors since there are scenarios where the momentum and mean-reversion investors do not submit orders at all.

The RL agent is able to maintain its relative good performance in terms of reward and inventory management while the random and the persistent agents absorb most of the market orders and inventory risk that the adaptive and RL agents passed on.

	Random	Persistent	Adaptive	RL
hedge cost	-0.0025	-0.0000	-0.0011	-0.0003
spread P&L	0.3426	0.1969	0.0039	0.0031
inventory P&L	2.8639	-2.8564	0.0386	0.0609
total P&L	3.2040	-2.6596	0.0415	0.0637
reward	-6.2054	-6.0884	-0.2364	-0.1487

Table 11: All 4 agents performance per time-step

	Random	RL
hedge cost	-0.0013	-0.0002
spread P&L	0.5179	0.0787
inventory P&L	0.0173	-0.0040
total P&L	0.5338	0.0745
reward	-0.5466	0.0032

Table 12: RL agent vs Random agent performance per time-step

	Adaptive	RL
hedge cost	-0.0037	-0.0004
spread P&L	0.5008	0.2118
inventory P&L	0.2015	-0.0247
total P&L	0.6986	0.1868
reward	-0.7590	0.0840

Table 14: RL agent vs Adaptive agent performance per time-step

If we compare the performance of the RL agent in all of the 1-vs-1 scenarios in this model with its performance in the previous model, we can see that relatively the RL agent is performing even better than it does before. The source of improvement could again be the ability of RL-based agent to manage inventory risk. If a market where less spread P&L pie is available for all agents to share, the RL-based agent stands out by balancing earning spread P&L and controlling the size of inventory.

	Persistent	RL
hedge cost	-0.0002	-0.0006
spread P&L	0.6879	0.0102
inventory P&L	-0.1638	0.1720
total P&L	0.5239	0.1816
reward	-0.3973	-0.3939

Table 13: RL agent vs Persistent agent performance per time-step

3.3 Addition of Informed Investors

When replacing some of the above 3 types of investor agents with informed investor agents, the magnitude of P&L for almost all the market making agents go down. It may be due to the fact that the existence of informed investors decreases the profit margin of market makers.

The RL agent still remains good performance because it has the highest reward and it keeps its inventory P&L very low. The conclusion that the RL agent performs well in inventory still holds. Meanwhile, the random agent and persistent agent take most of the market orders, making their inventory level and inventory P&L very high, which is shown in Figure 18 and Figure 20.

	Random	Persistent	Adaptive	RL
hedge cost	-0.0030	-0.0000	-0.0008	-0.0001
spread P&L	0.0554	0.3071	0.0011	0.0003
inventory P&L	2.0077	-1.8276	0.0066	0.0009
total P&L	2.0601	-1.5206	0.0069	0.0011
reward	-5.1473	-4.3643	-0.0773	-0.0073

Table 15: All 4 agents performance per time-step

	Random	RL
hedge cost	-0.0013	-0.0002
spread P&L	0.4243	0.1466
inventory P&L	0.0031	-0.0061
total P&L	0.4261	0.1403
reward	-0.5197	0.0688

Table 16: RL agent vs Random agent performance per time-step

	Persistent	RL
hedge cost	-0.0003	-0.0003
spread P&L	0.3859	0.0041
inventory P&L	-0.0298	0.0211
total P&L	0.3557	0.0250
reward	-0.5574	-0.0946

Table 17: RL agent vs Persistent agent performance per time-step

	Adaptive	RL
hedge cost	-0.0018	-0.0003
spread P&L	0.4243	0.1534
inventory P&L	-0.5594	-0.0495
total P&L	-0.1369	0.1037
reward	-0.5540	0.0373

Table 18: RL agent vs Adaptive agent performance per time-step

In 1-vs-1 scenarios, all the P&L values decrease. The RL agent remains excellent performance thanks to inventory management. In the cases of RL vs Random/Persistent agents, RL agent's performance improves compared with the corresponding cases in the previous model. The existence of informed investors hurt the naive structured market makers while RL can still relatively maintain its performance form the previous model. And in the case of RL vs Adaptive agent, the RL agent even wins in total P&L according to Figure 11. Still, we credit

the robust performance to the strong inventory management capability of the RL agent as we can see in Figure 10 and Figure 12.

4 Conclusions and Future Work

To conclude our project report, we believe that we have achieved a lot with this project. By building on the idea and model specification introduced by the Ganesh paper, we built a comprehensive programmatic environment where we can simulate the trading of a dealer's market while allowing easy implementations of any types of abstractions to the agents participating in the market.

Although our dataset differs from the dataset used by the paper we referenced to and our dataset lacks certain important elements, we do manage to find ways to establish connections between our simulated environment and the real data. Specifically, we are able to fit a reference spread curve using the LOB data of our dataset. We also find a way to make our mid-price process close to the real price process in our market data by varying the input parameters for our random agents and matching the signal-to-noise ratio.

Besides matching our simulated environment to the dataset we collected, we make further attempts to make our simulations realistic by introducing investor agents with realistic behavior. These investor agents correspond to real-life investors with certain investment schemes and asymmetric information.

In the results section, we find that although our RL-based agent do not outperform the other agents in terms of PL but it does successfully achieve a balance between earning spread PL and managing inventory risk as it is trained to do while the other agents are not able to maintain such balance. In a real-world setting, this balancing behavior of our RL agent can be argued to be more desirable as a market-maker typically do not wish to maintain a large inventory over a long span of time. We do not, however, find evidence that the RL agent is able to learn the strategies of other agents. This is especially evident in the fact that in a 1-vs-1 situation between the RL agent and the persistent agent, the RL agent could easily

win more trades and earn more spread P&L by beating the persistent agent's spread by a tiny margin. However, the results show that the RL agent fails to recognize this as a feasible way to earn profit.

Just like every other project, there is always more that can be done in the future with our project topic. Like mentioned in the previous sections, our dataset does not include trading data from the dealer's market which makes comparing and matching statistics of trade frequencies impossible. This can be feasible in the future given that the right type of dataset is provided.

Currently, we have assumed a simple model for lots of aspects of the simulation, from the behavior of the different agents to the way reference spread is constructed. By combining a dataset with larger size and more sophisticated model, we could potentially further improve the realness of our simulations.

Finally, to address the problem that the RL agent cannot learn the other agents' strategies, in the future, it would be worthwhile to abstract behaviors of other agents as input to the RL agent's training process. We believe that this could be key to train an RL agent that can adapt to other agents' strategies which in our project the RL agent fails to do because of the relatively naive RL training.

References

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques forsimulating human systems.

Ganesh, S., Vadori, N., Xu, M., Zheng, H., Reddy, P., & Veloso, M. (2019). Reinforcement learning for market making in a multi-agent dealer market.

Appendices

A Simulation Environment Parameters Settings

Parameter	Value
μ (annualized GBM mid-price drift)	0.20
σ (annualized GBM mid-price volatility)	1.05
mid-price	75
PIC	0.3

Table 19: Parameters for Simulation Environment

	ϵ_{min}	ϵ_{max}
20 noise agents	-0.10	0.10
10 noise + 5 momentum + 5 mean-reverse agents	-0.46	0.28
8 noise + 4 momentum + 4 mean-reverse + 4 informed agents	-0.19	0.46

Table 20: Parameters for Random Agent

	ϵ^b	ϵ^a
20 noise agents	0.5	-0.5
10 noise + 5 momentum + 5 mean-reverse agents	0.13	-0.03
8 noise + 4 momentum + 4 mean-reverse + 4 informed agents	-0.03	-0.50

Table 21: Parameters for Persistent Agent

B Graphs for Experiment: Addition of Informed Investors

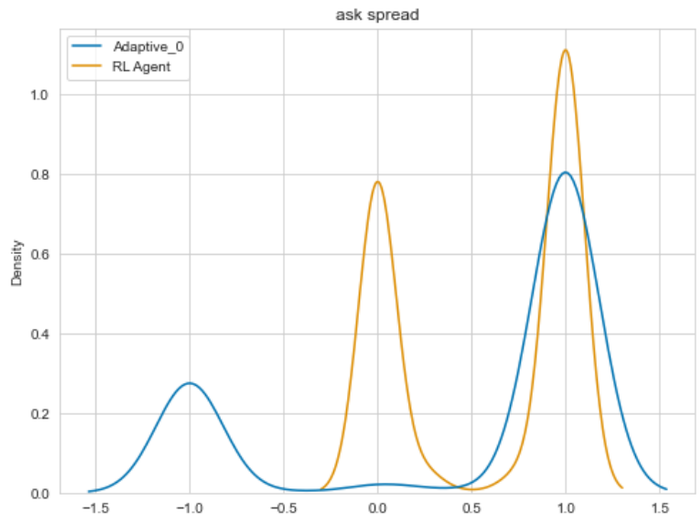

Figure 7: Ask spread distribution: RL agent vs Adaptive agent

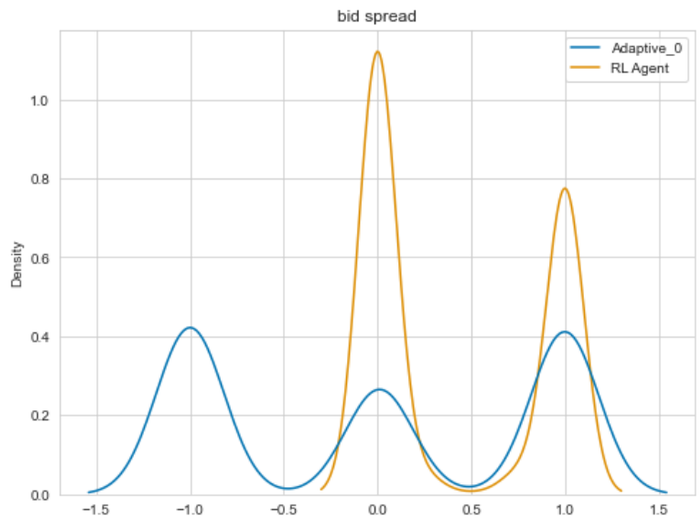

Figure 8: Bid spread distribution: RL agent vs Adaptive agent

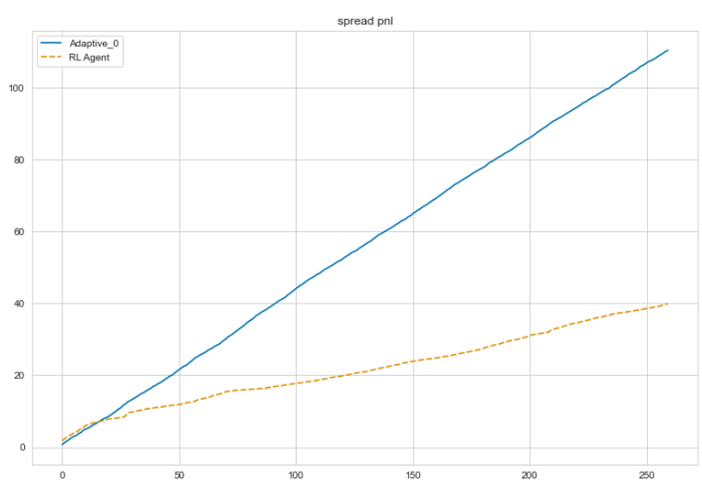

Figure 9: Cumulative spread P&L: RL agent vs Adaptive agent

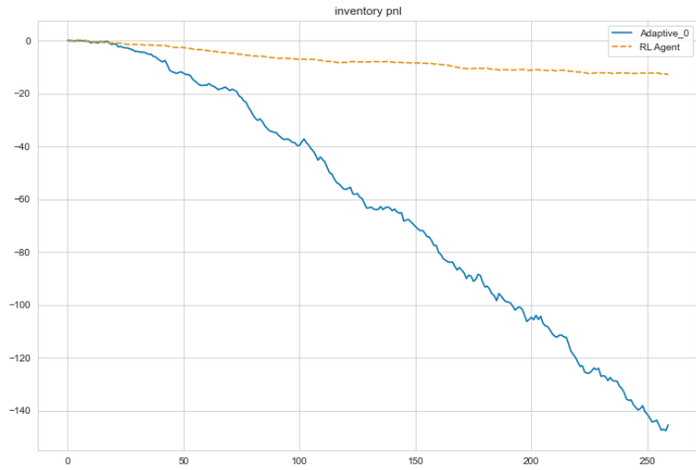

Figure 10: Cumulative inventory P&L: RL agent vs Adaptive agent

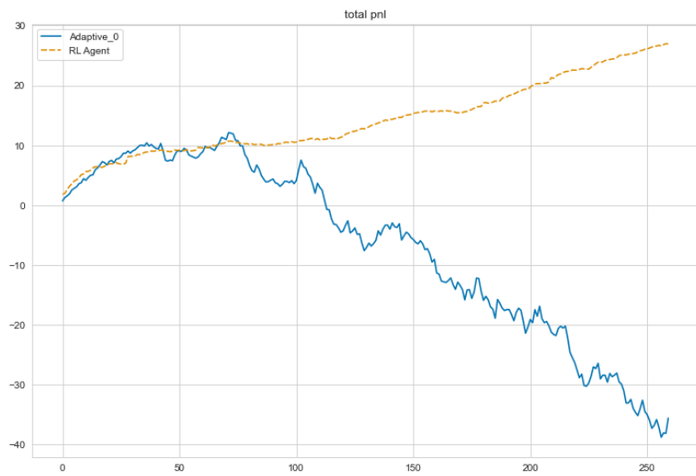

Figure 11: Cumulative total P&L: RL agent vs Adaptive agent

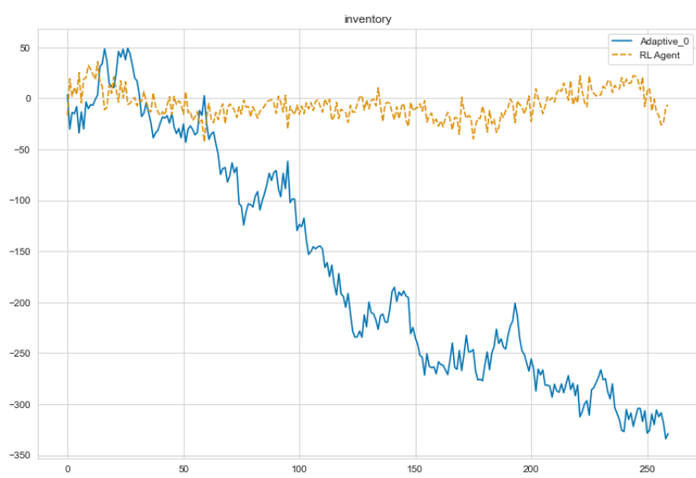

Figure 12: Inventory: RL agent vs Adaptive agent

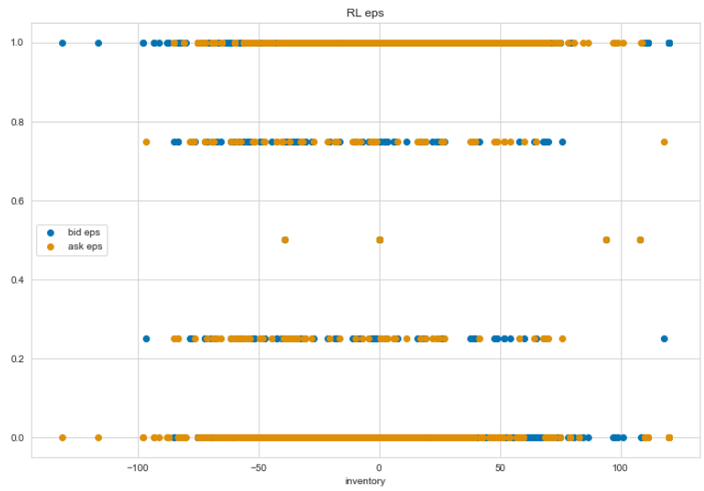

Figure 13: Inventory vs ϵ: RL agent vs Adaptive agent

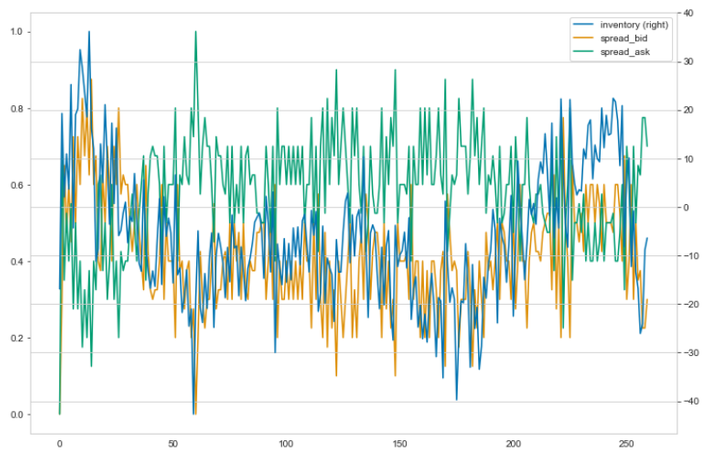

Figure 14: Inventory vs ϵ time series: RL agent vs Adaptive agent

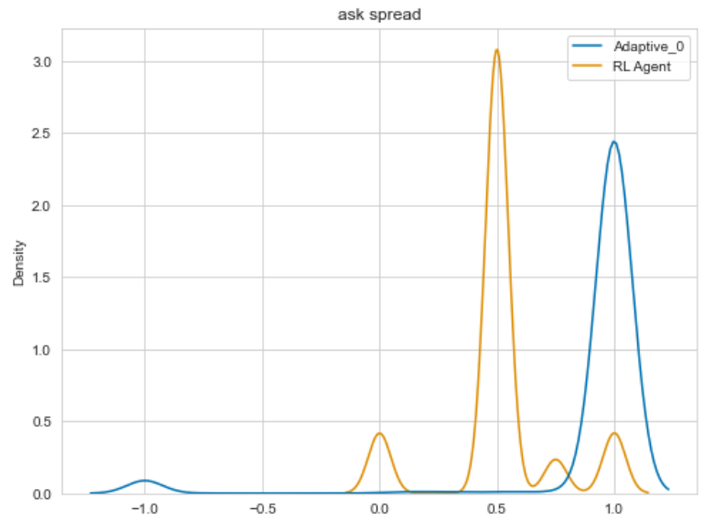

Figure 15: Ask spread distribution: All 4 agents

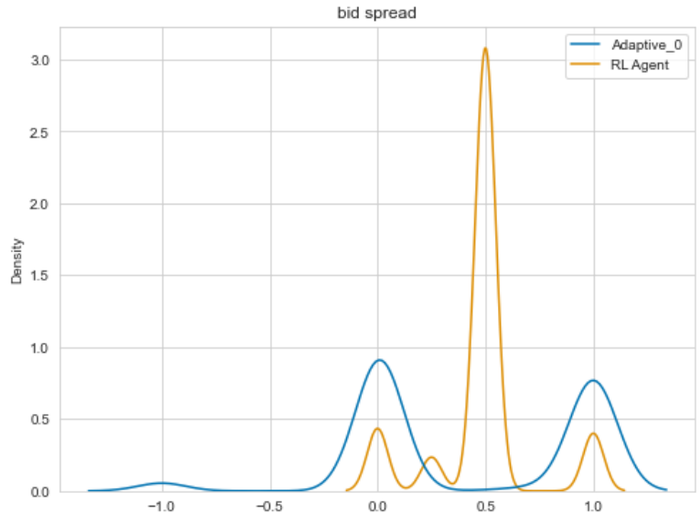

Figure 16: Bid spread distribution: All 4 agents

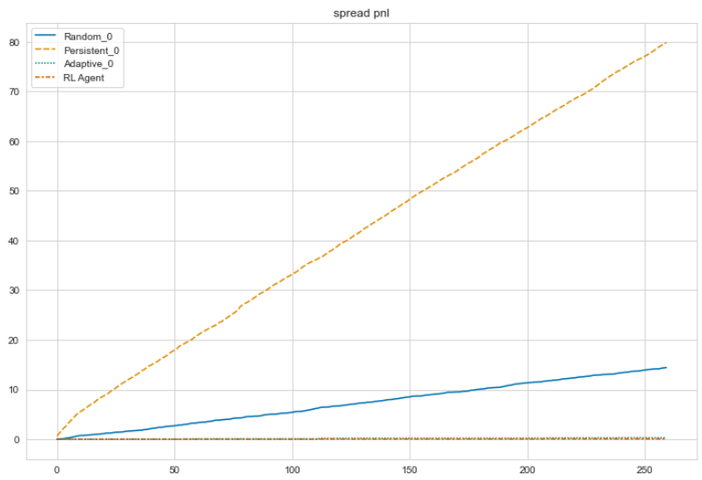

Figure 17: Cumulative spread P&L: All 4 agents

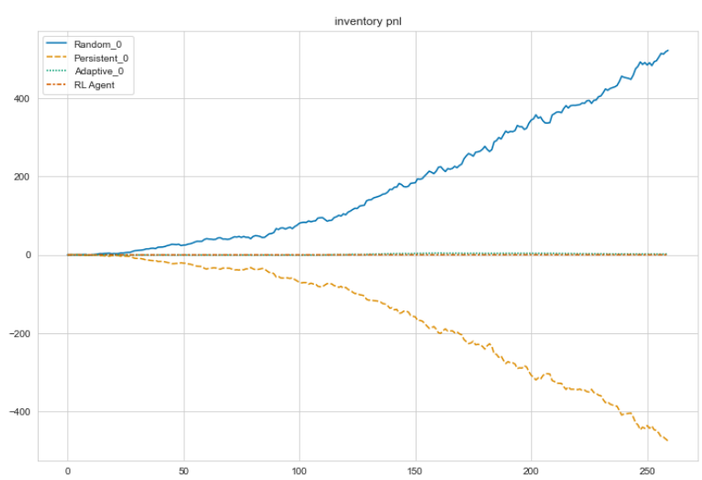

Figure 18: Cumulative inventory P&L: All 4 agents

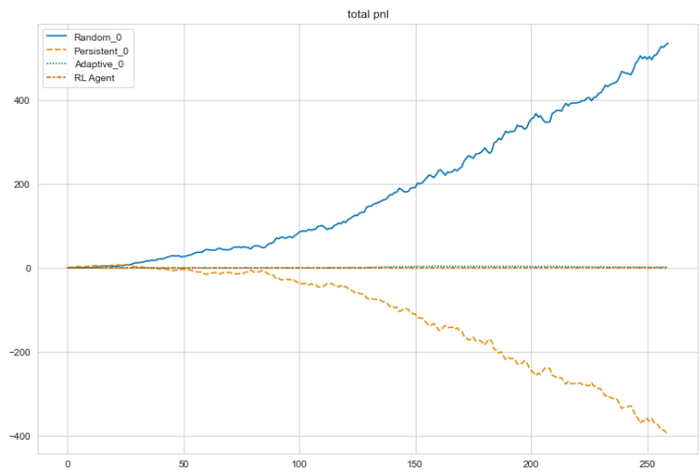

Figure 19: Cumulative total P&L: All 4 agents

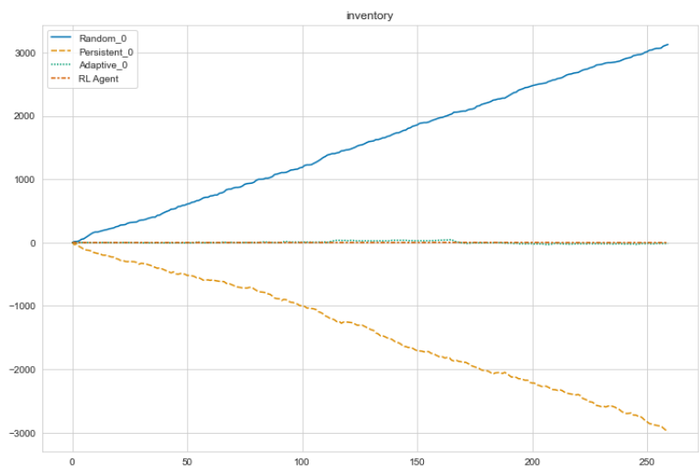

Figure 20: Inventory: All 4 agents

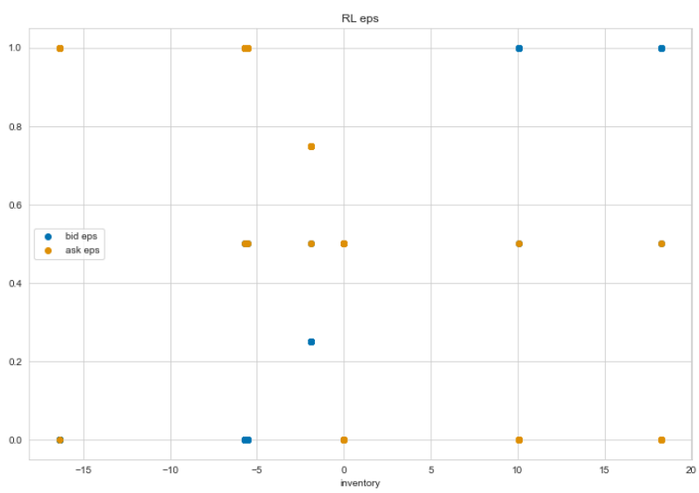

Figure 21: Inventory vs ϵ: All 4 agents

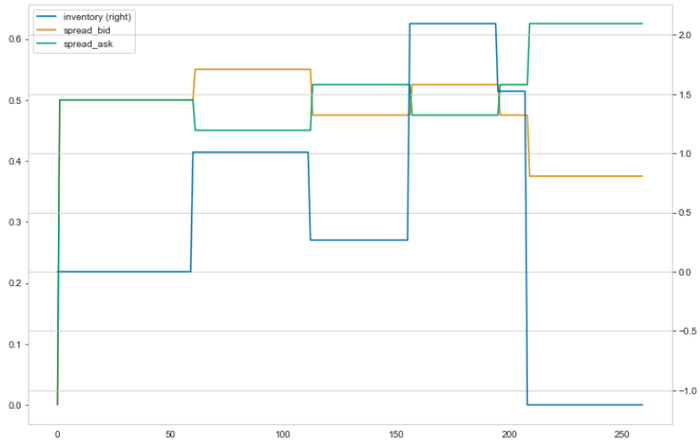

Figure 22: Inventory vs ϵ time series: All 4 agents