Market Making in the High-Frequency Trading Limit
CS234 Project Paper

Ben Kroul [1] **David G. Nuñez** [1] **Ryan Padnis** [1]

Abstract

Market makers are one of the biggest providers of liquidity in financial markets. Whether they are dealing with stocks, bonds, options, or even exotics, market making has become a mainstream way for traders to exploit market inefficiency and create statistical arbitrage. In addition to demystifying this field, we aim to tackle this challenging problem using a reinforcement learning agent. While past reinforcement learning market making agents have used a host of assumptions that decouples market dynamics from the agent's actions, we implement a direct limit-order book simulation method that is more realistic for a real-time high-frequency trading setting.

1. Introduction

1.1. Overview

We begin by first defining the terminology essential to our model. We will be working in a high frequency environment, using millisecond-by-millisecond data, under the assumption of no latency or transaction costs associated with our electronic market. Said market primarily consists of a limit order book wherein we distinguish between limit and market orders.

This limit order book is a listing of all possible bids (offers to buy) and asks (offers to sell) of a stock at a specified price and quantity (see Fig. 1). Each limit order stays in the book until either someone on the opposing side decides to execute it by placing a market order or we as the market maker remove it ourselves.

A market order is an immediate purchase, or sale, of a security at the best price available. This could either refer to the best/highest bid-price or the best/lowest ask-price, depending on whether the order is sell or buy, respectively. Placers of market orders are market takers; they simply ask for a quantity and are given the securities at one of more prices depending on their execution amount. If the quantity specified by that limit order runs out, then the rest of the market order "eats" the next best ask price, and so on.

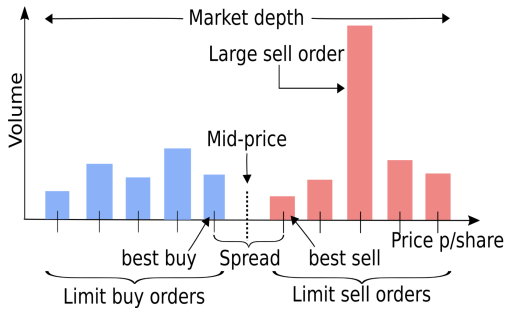

Figure 1. Basic model of a limit-order book

As a market maker, our goal is to constantly reshape the order book with limit orders to convince potential market participants to engage, thus allowing us to accumulate wealth over time. We work in a large volume, high frequency environment to explore an area of market making that has not been well researched in a reinforcement learning perspective.

While market making is usually optimized from a purely statistical background using statistical arbitrage (Xiong et al., 2015; Law & Viens, 2020), we want to use reinforcement learning to optimize a market making agent and find multiple possible valid market making strategies. Reinforcement learning has been used before to optimize various similar problems, such as a statistical arbitrage problem on high-frequency trading by Lim & Gorse (2018), high-frequency market making strategies using a Gaussian-kernel approximation of a limit order book by Gasperov & Kostanjcar (2022), and using historical data to model a functional approximation of limit order book by Guo et al. (2023). While all of these implementations have merit as approximations under their respective assumptions, they effectively decouple the market maker's actions from the market's trading

[1]Stanford University, Stanford, California. Correspondence to: Ben Kroul <benkroul@stanford.edu>, David G. Nuñez <davidgn@stanford.edu>, Ryan Padnis <rnpadnis@stanford.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

intensity. We realize direct agent-market coupling via implementing a fully-functional limit order book that is used to simulate market trajectories.

1.2. State Space

We define our environment in accordance with the landmark paper on high-frequency market making written by Avellaneda & Stoikov (2008). We denote S_t as the mid-price of the stock at time t which is meant to represent the average between the best bid price and the best ask price across all markets. The mid-price is a good theoretical expectation for the average value of a stock at time of liquidation, and so it will solely be used to evaluate the stock price at the end of each simulated episode. At high frequency and without access to all markets and limit order books containing a given stock, the mid-price is not actually the arithmetic average, so we instead utilize the following Brownian motion without drift Stochastic Differential Equation (SDE) to simulate the discrete updates to the mid-price:

$$dS_t = \sigma dW_t \tag{1}$$

The use of this specific stochastic update is simply that on these time-scales—millisecond steps and trajectories that don't even extend out to ten minutes—perturbations to the mid-price are well-approximated by the limit of a symmetric random walk. We also define $\delta^b = S_t - p^b$ as the bid spread and $\delta^a = p^a - S_t$ as the ask spread. We expect that in an efficient market, makers and takers act to functionally restore spreads to an optimum, so when they are large, our agent should learn that closing the spread likely benefits us in the long run, and when they are small, it should widen them to maximize wealth 'transfer'.

Based on the number of market orders we have received in a period, our stock holdings/inventory will also be stochastic variable. We denote the market maker's inventory as $I_t = N_t^b - N_t^a$ where N_t^b is the process governing the cumulative number of stocks we have bought via other simulated agents' market sell orders, and vice versa with N_t^a (in the sense we are effectively taking on a short position as the counterparty to a market buy). In (Avellaneda & Stoikov, 2008), they find that market order intensities should follow a Poisson process with rates $\lambda^b(\delta^b), \lambda^a(\delta^a)$ that are each parameterized by an exponential function.

Finally, we also observe our wealth W_t which is a function of the change in inventory components and price at the time of the market order. We govern this process by the following SDE:

$$dW_t = p^a dN_t^a - p^b dN_t^b \tag{2}$$

To track all of the data in the order book, as opposed to only tracking the best bids and asks, we begin by defining an `OrderBook` class that contains the full limit-order book – even so-called "stub quotes" far from best bid/ask not meant for immediate execution – and can be used to return these parameters and the entire distribution of limit orders.

So in full, our observation at time t is

$$O_t \equiv \left(W, I, S, \delta^b, N^b, \delta^a, N^a\right)_t \tag{3}$$

which corresponds to observing our internal state of wealth and inventory along with the state of the LOB. We also include t in the observation to account for temporal correlations.

1.3. Action Space

In the real world, market makers are sampling their states with some latency from an environment that contains a whole lot of different agents: value investors, arbitrage investors, other high-frequency market makers, wealth investors, etc. They can't control this but they can exert influence over a sufficiently large proportion of the limit order book so as to effectively achieve total influence within the scope that a limit order book *can* influence other agents, i.e. within the market they've made. As such, in our environment, we too seek to exploit inefficiencies in the processes simulating other-agent behaviors solely through our control of the limit order book. We can place both bid and ask limit orders:

$$\boldsymbol{a}_t = (P^b, N^b, P^a, N^a)_t. \tag{4}$$

On our short time-scales, we ideally only submit limit orders that change the new best bid or ask. This could result in either increasing or decreasing the spreads depending on the sign of N, with a negative number of stocks corresponding to increasing the spread by removing a bid or an ask. On longer time-scales, we must also make sure that there are always stocks in the limit-order book, and may add limit orders at a larger spread to account for this.

1.4. Reward Function

We first considered using a 'frozen reward' similar to Avellaneda & Stoikov (2008) such that the intermediate reward is discounted by the time left,

$$\exp(-\gamma' \tau (W_T + I_T S_T)), \tag{5}$$

where γ' is the constant absolute risk aversion and τ is the time left until the terminal time.

We however found that while this for many choices of γ this function would have trouble learning what to do before the terminal time period, and thus the learning was erratic.

We therefore introduce a final reward,

$$W_T + I_T S_T, \tag{6}$$

that is only given to the agent at the terminal time T and is then discounted by a *discount factor* γ that propagates backwards when computing the trajectory returns which will incentivize the final liquidation "value" of the market.

To stress the order book providing enough liquidity to the market during intermediate times, we decided to only give the reward if the order book does not run out of limit orders before the terminal time (early termination). By disincentivizing early termination we force our agent to escape the local minima of getting rid of all its inventory as fast as possible and being able to prioritize long term rewards with an understanding of the sustained dynamics of full trajectories of the simulation.

Another reward we considered is from a recent paper that introduces an immediate utility based reward that can be discounted to further improve the model over times that are farther away from the terminal time (Lim & Gorse, 2018). They introduce an immediate reward function parameterized by the wealth difference dW_t, inventory difference dI_t, time remaining in the horizon $\tau_t = T - t$ (for terminal time T), and two positive constants a and b:

$$R_t = a(dW_t) + e^{-b\tau_t}\text{sgn}(dI_t) \tag{7}$$

This will allow us to learn intermediate rewards that will also help us improve total value over time. However, the papers utilizing these reward functions simplify the action space to solely learning the spreads, not the long term repercussions of the quantities associated with submission of limit orders. We found that this intermediate reward function did not properly incentivize long times, so we define a full intermediate reward function of

$$R_t = a(dW_t) + e^{-b\tau_t}\text{sgn}(dI_t) + c(T-t) \tag{8}$$

and decide to normalize a, b, c so that each element has a realistic maximum of 1. This reward ensures that our agent is taking both long and short positions and is reaching the terminal time (see section 3).

1.5. Parameter Estimation

To build a consistent dynamics model, we gathered high frequency data from Wharton Research Data Services of a day's worth of market book movements for the S&P500, an index fund containing 500 of the largest valued public companies.

We model the mid-price by sampling Brownian motion. To infer the drift and scale parameters of said Brownian motion, we back out MLEs of the discrete time approximation for the following SDE:

$$dS_t = \mu dt + \sigma dZ_t \tag{9}$$

Where Z_t is a standard brownian motion. We use the paper from (Mykland & Zhang, 2009) for our inference and obtained the parameters $\widehat{\mu} = 3.59 * 10^{-6}, \widehat{\sigma} = 2.4 * 10^{-3}$.

We make the well-founded assumption that the arrivals of market orders follows a Poisson process. The mean parameter of these processes, λ_a and λ_b, can be expressed as the following function of q, the current quantity associated with the best bid or ask,

$$\lambda(q) = \exp\left(\beta_0 + \beta_1 \ln(1+q) + \beta_2(\ln(1+q))^2\right), \tag{10}$$

as described in (Toke & Yoshida, 2017). Running linear regression on the log of this equation, we obtain unbiased estimation on the parameters. However, we immediately found that the estimation was inconsistent and would lead to the order book terminating too quickly as our agent would learn an unrealistic situation. As a result, we decided to implement the spread into this equation as well, denoted as s.

$$\begin{aligned}\lambda(s,q) = \exp\Big(&\beta_0 + \beta_1 \ln(1+q) + \beta_2(\ln(1+q))^2 \\ &+\beta_3 \ln(1+s) + \beta_4(\ln(1+s))^2 \\ &+\beta_5 \ln(1+s+q)\Big)\end{aligned} \tag{11}$$

The interpretation of this function for the intensity is that it acts as a demand function. When the best quotes have high quantity, this indicates to the consumer the supply is large, and so they will demand less as many of the consumers will wait for a better price. Additionally if the spread is too large, then the demand will be small, forcing our agent to shrink the spread. This creates a nice push-pull dynamic between the agent and the market.

2. Reinforcement Learning Approach

To learn an incredibly complicated environment, we use policy gradients to easily leverage neural networks to model the high dimensional complexity of our action space and stochastic policies for the optimal policy class. Previous work in Guo et al. (2023) implements a highly complex neural network infrastructure for both the policy and value approximation that takes in many observation, action, reward pairs using multiple convolutional and attention layers. Due to time constraints, we implement a much simpler architecture that consists of 2 hidden layers of size 10, with `ReLU` activations.

2.1. Code layout

2.1.1. LIMIT ORDER BOOK

Starting with the most novel thing about our approach to market making with reinforcement learning on limit order books, we *actually implement* a limit order book, whereas most in the literature just work around actually implementing and interacting with a simulated one. The file `book.py` contains the `LimitOrderBook` class, which represents a limit order book tracking bids and asks. This class has attributes such as `bids` & `asks`. The methods in this class include `add_order(side, price, quantity)` to add an order to the book, `remove_order(side, price, quantity)` to remove an order, and `get_best_bid()` & `get_best_ask()`.

2.1.2. RL & MM CONFIGURATION

Still detailing the infrastructure on top of which our model learns and trains, `config.py` file defines the `Config` class, which holds configuration settings for the market making simulation. This class includes attributes such as `trajectory` that controls returns type, `n_episodes`, `n_batches`, `n_epochs`, `do_ppo`,lambda, etc. Basically every parameter that we've varied in our experiments to tune said parameters to result in the best policy.

2.1.3. MARKET ENVIRONMENT

The `Market` class in `market.py` creates a detailed market environment using a limit order book, simulating the actual trading environment where the market maker operates. This class is initialized with a configuration object and maintains an instance of `LimitOrderBook` to manage the bid and ask orders. The `reset()` method reinitializes the order book and the time step, setting up a fresh environment for each new episode of training or testing. The `step` method is central to the simulation, evolving the market order book by updating the midprice and placing market orders. It computes changes in wealth and inventory (`dW` and `dI`), as well as tracking the number of bid hits and ask lifts. The `lambda_buy` and `lambda_sell` methods model the arrival rates of buy and sell market orders based on the current state of the order book and specified betas. The `state` method returns the current state of the market as a tuple, including the number of bids and asks and their respective prices. The `act` method determines the actions to be taken on the order book, which can be based on a naive policy, a policy network, or an Avellaneda-Stoikov strategy that adjusts prices based on the reservation price and optimal spread. Actions determined by the `act` method are executed using the `submit` method, which places limit orders on the LOB. This method ensures the bid and ask deltas are non-negative and rounds the order quantities to integers. The market dynamics and trading environment created by the `Market` class are crucial for training and evaluating the performance of market making policies, enabling realistic simulations of trading strategies. This comprehensive environment setup is crucial for accurately simulating market conditions, allowing the market making policy to learn and adapt effectively through reinforcement learning. We also have `fast_market.py` with the `FastMarket` class, which tries to leverage an all-PyTorch approach for speed and running everything possible on the GPU.

2.1.4. TRAINING THE MARKET MAKER CODE

The `MarketMaker` class integrates the `Market` environment with a reinforcement learning policy. It initializes the market and policy using the given configuration and provides methods for training and testing the policy. The `train` method runs multiple episodes, resetting the market each time and using the policy to select actions and update Q-values based on observed rewards. The `test` method evaluates the policy in the market environment, tracking the total reward over the episode. The `save` and `load` methods manage the persistence of trained models, saving and loading policy networks and final returns. The `get_paths` method generates trajectories and computes rewards based on the immediate state, supporting both standard and PPO (Proximal Policy Optimization) training. It handles variable-length trajectories and computes advantages for policy updates. The `plot` methods visualize the final scores and trajectories, providing insights into the performance and behavior of the trained market making policy. The `Policy` class in `policy.py` represents the reinforcement learning policy used by the market maker. We mainly utilize a Gaussian Policy like the one from Assignment 2, and didn't end up implementing the discrete Categorial policy.

2.1.5. MISCELLANEOUS AND NOT SO MISCELLANEOUS

In `rewards.py`, the function `calculate_reward(inventory, cash, price, config)` calculates the reward based on the current inventory, cash, and price, returning a reward scaled by the reward scaling factor in the configuration. There are actually several different reward functions contained here, as a good reward function was really one of biggest chokepoints for this project. Finally, the `util.py` file provides all of the odds and ends, many of which are very important for actually plotting the visualizations of our training process, that allow everything to gel and flow.

2.2. Return Algorithm

We consider Monte-Carlo returns (`MC`) and eligibility trace returns (`TD`).

For a while, we were thinking of fully incorporating TD

learning, using the past x states to inform the current action instead of just the current state.

However, due to time constraints, we decided to settle for just taking into account the current state to keep the algorithm space simple and let trainings take a reasonable amount of time.

3. Experimental Results

To start with a sufficiently large order book, we simulate 0.3 seconds (300 time steps) of the environment transitions with our policy being submitting limit orders at random. This provided the best consistency fro initial state distirbution and also yielded the msot impressive results. Below we include a sample series of plots under the benchmark policy.

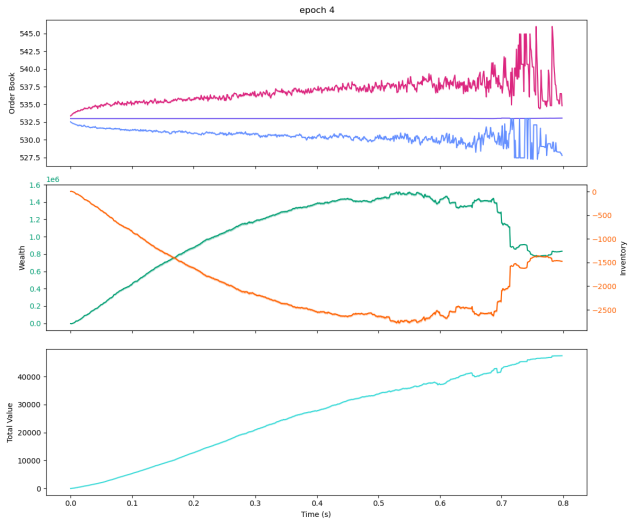

Figure 2. The action of the default policy over 100 trajectories. The top plot shows the evolution of the lowest ask P_a (red), highest bid P_b (blue), and the midprice S_t(purple). The middle plot shows the cumulative wealth W_t and inventory I_t. The bottom plot shows the total intermediate market value of our cummulative market value, $W_t + I_t S_t$

We found an optimal number of timesteps of 5000, corresponding to a terminal time of 5s, and an optimal number of batches of 100. Ideally the number of batches per epoch should be higher to fully sample the Brownian motion of the order book midprices, but due to time constraints we wanted training that took less than 4 hours to complete per policy.

3.1. Determining the Best Reward Function

We consider two primary ways of incentivizing the agent to reach terminal times.

The first is by giving it an intermediate time reward that increases for longer times in accordance with equation 8, which we will call the `immediate` reward strategy.

The second is letting the market liquidate its assets regardless of reaching the terminal time, with a thought that it will linearly increase the final value of $W_T + I_T S_T$. Let's call this the `liquid` reward strategy.

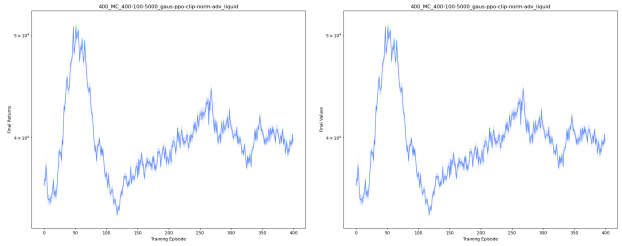

Figure 3. The performance of the `liquid` reward strategy over time as measured by the final scores (left) and final values (right) of the market maker. Note that the returns are equivalent to the plotted values for this reward strategy.

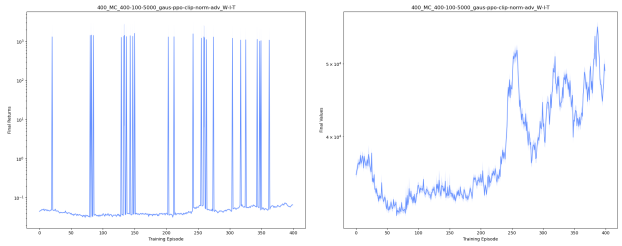

Figure 4. The performance of the `immediate` reward strategy using `MC` returns, as measured by the final scores (left) and the final values (right). This is our best-performing case (see Fig. 13)

Both of the above reward strategies use PPO, but clearly the `liquid` reward function learns a much better policy. In figure 3, we see that while the policy gets worse after reaching its maxima in final values, it finds this after relatively few training episodes and is on the same order of magnitude as, say, the final values of `immediate` rewards with `MC` returns.

We also tried adding an immediate reward that was just dW_t. The results are shown below:

3.2. Determining the Best Returns Method

We either use default Monte-Carlo returns, `MC`, or implement the TD-λ eligibility trace, `TD`.

The performance of `TD` is significantly decoupled from the

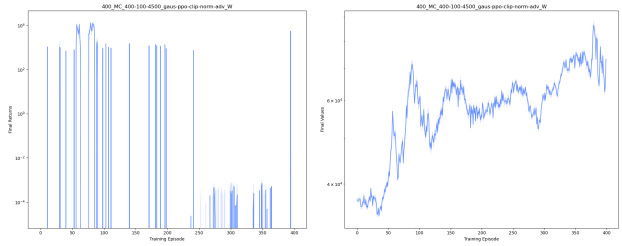

Figure 5. `MC` returns with only a dW_t reward, using `PPO`

values we want to see for our reward functions, with value plots decreasing across epochs. The `MC` returns show improvement over epochs, which is what we want to see. It is a future question of why `TD` returns doesn't generate the correct value.

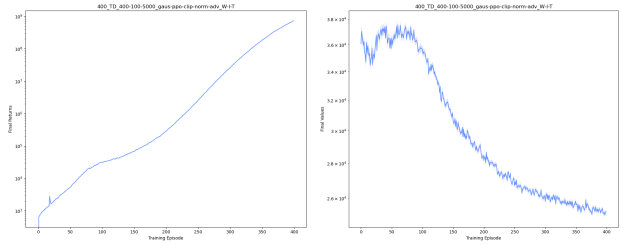

Figure 6. The performance of the `immediate` reward strategy over time using `TD` returns, both the internal final score (left) and the final values (right).

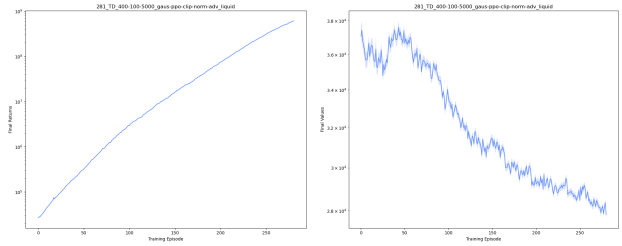

Figure 7. The performance of the `liquid` reward strategy over time using `TD` returns, both the internal final score (left) and the final values (right).

3.3. Removing Parts of PPO one at a time

We wanted to explore whether the complete PPO gradient method is really necessary, so let's build up to PPO from just using REINFORCE. The results are not too surprising. All of the following runs are made using Monte-Carlo returns and an intermediate reward function.

3.4. Fine-Tuning Our Best Policy

We found that using `MC` returns and `intermediate` rewards provided the best optimization with our desired mar-

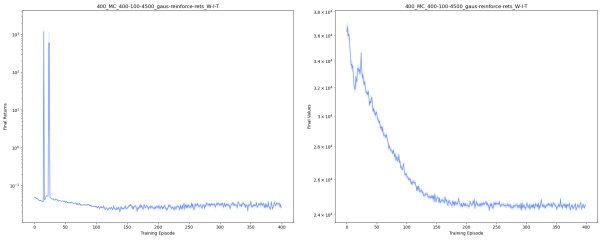

Figure 8. The performance of REINFORCE without using an advantage function.

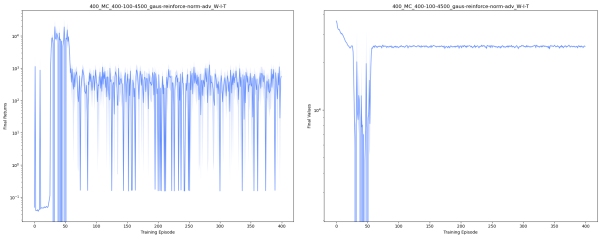

Figure 9. The performance of REINFORCE with a normalized advantage function.

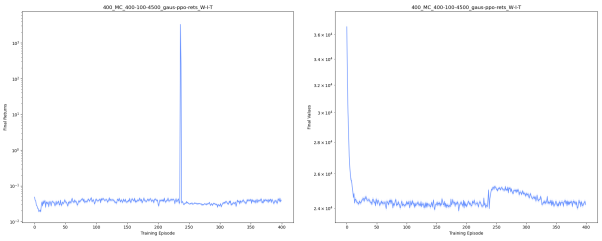

Figure 10. The performance of PPO without clipping and without an advantage function.

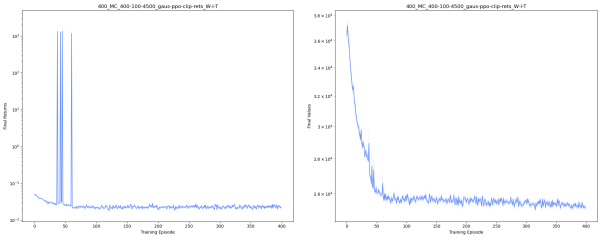

Figure 11. The performance of PPO with clipping but without an advantage function.

ket behavior. Using `PPO`, with a normalized advantage function and with clipping, proves to be way more robust than simpler policy gradient methods. We therefore combine these aspects and fine-time our reward function in order to achieve our best run:

Between these epochs, we can see that our optimal policy learns to prioritize the final liquidation more and more. We

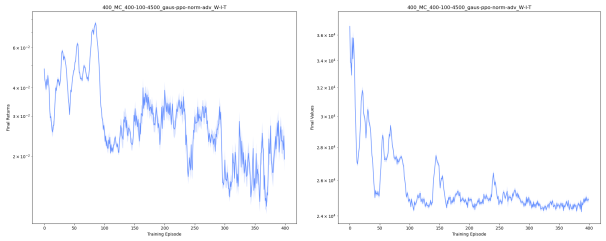

Figure 12. The performance of PPO with an advantage function but without clipping.

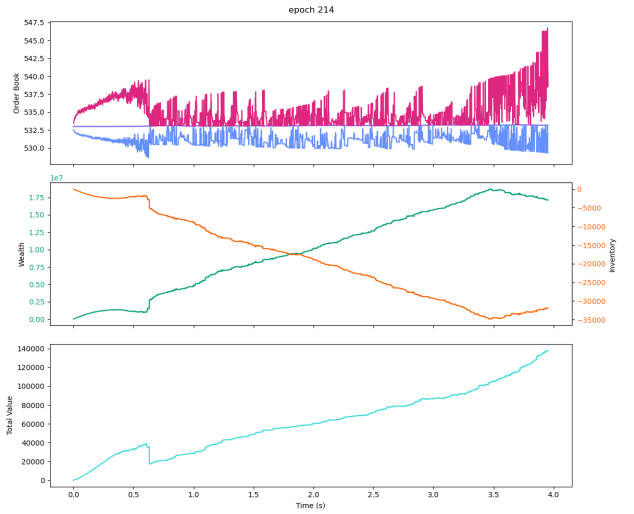

Figure 13. Our best run so far, maxed out at 215 training epochs with a final value of $120,000, over a single second! See Fig. 4

can observe this with the change in directions of the wealth and inventory processes, where the minimum/maximums occur earlier as the episodes progress. We see the order book stabilizes as well, where the best bids and asks both progress more consistently, indicating better convergence to an optimal policy. Towards the end, we see that there are heavy fluctuations. This is because the policy has learned to engage in tactics to draw consumers in towards the end so our inventory is large, giving us a great terminal reward. While the final reward is lower for this policy, we have learned a better allocation of resources that leads to a better trajectory on average, versus a high-risk high-reward strategy.

Finally, we can see that the order book ends up terminating much quicker. This is because the market making agent realizes that the market order agents will adapt quickly to the environment over longer periods of time. This is an interesting, yet expected byproduct of our construction, but in the end it may be a good idea to consider other type of penalties to prevent early termination.

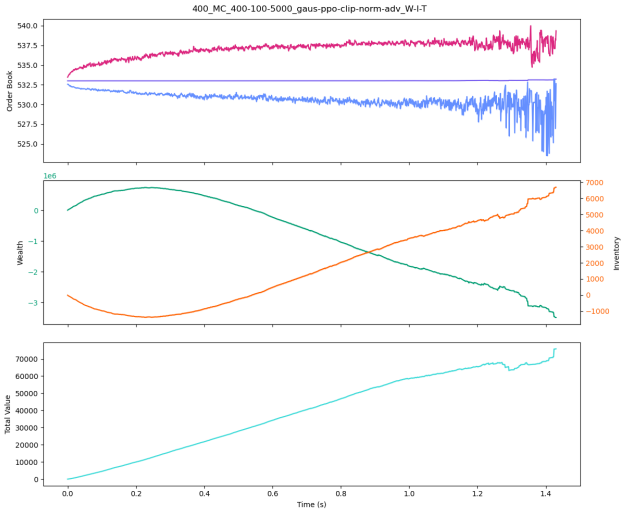

Figure 14. The best learned policy, after 400 training epochs. It is interesting that at first, the market begins taking a short position on the stock, and then ends with a long position on the stock.

4. Conclusion

4.1. Future Work

We have had to face a ton of issues throughout this project that can definitely be further fine tuned in the future. For starters, utilizing some of the more recent, cutting-edge literature on statistical techniques for simulating our limit order book dynamics. This could include changing the mid-price model to geometric Brownian motion, or using a different statistical estimation technique such as high frequency regression.

A better reward function really is the holy grail here, though, and the literature is full of attempts to derive them. We essentially had to resort to reward engineering ourselves to achieve consistent learning properties and minimize the amount of undesirable reward hacking occurring, but using imitation learning with an expert policy from realized data could help to create a more robust reward function.

We worry the current implementation of our overall model might not be incredibly conducive to generalization. Transaction costs and latency are actually some of the biggest problems faced by market makers. Including these would probably show that the market is not as arbitrageable as we have made it out to be, as even the most naive algorithms resulted in our market making agent profiting (albeit we see improvement from the initial to final agent). We did not have the appropriate resources, but in the future if possible using limit order book data to backtest this policy and our simulation would also allow us to speak more to the efficiency of these algorithms, rather than relying solely on our

possibly oversimplified simulation.

Finally, possibly introducing many competing market making agents could reduce the presence of arbitrage opportunities. This could include also introducing and training agent for the market order dynamics, or even a market agent for the mid-price dynamics. In reality, in large volume market making there are so many different players that affect the order book, properly figuring out a way to model any more or all of these agents would improve our accuracy, and in some way we're contributing part of that complex ecosystem with the market making agent itself.

References

Avellaneda, M. and Stoikov, S. High-frequency trading in a limit order book. *Quantitative Finance*, 8(3):217–224, 2008. doi: 10.1080/14697680701381228. URL https://doi.org/10.1080/14697680701381228.

Gasperov, B. and Kostanjcar, Z. Deep reinforcement learning for market making under a hawkes process-based limit order book model. *IEEE Control Systems Letters*, 6:2485–2490, 2022. ISSN 2475-1456. doi: 10.1109/lcsys.2022.3166446. URL http://dx.doi.org/10.1109/LCSYS.2022.3166446.

Guo, H., Lin, J., and Huang, F. Market making with deep reinforcement learning from limit order books. 2023. URL https://arxiv.org/abs/2305.15821.

Law, B. and Viens, F. Market making under a weakly consistent limit order book model. 2020. URL https://arxiv.org/abs/1903.07222.

Lim, Y.-S. and Gorse, D. Reinforcement learning for high-frequency market making. In *The European Symposium on Artificial Neural Networks*, 2018. URL https://api.semanticscholar.org/CorpusID:53244064.

Mykland, P. A. and Zhang, L. Inference for continuous semimartingales observed at high frequency. *Econometrica*, 77(5):1403–1445, 2009. doi: https://doi.org/10.3982/ECTA7417. URL https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA7417.

Toke, I. M. and Yoshida, N. Modelling intensities of order flows in a limit order book. *Quantitative Finance*, 17(5):683–701, 2017. doi: 10.1080/14697688.2016.1236210. URL https://doi.org/10.1080/14697688.2016.1236210.

Xiong, Y., Yamada, T., and Terano, T. Exploring market making strategy for high frequency trading: An agent-based approach. In Takayasu, H., Ito, N., Noda, I., and Takayasu, M. (eds.), *Proceedings of the International Conference on Social Modeling and Simulation, plus Econophysics Colloquium 2014*, pp. 63–74, Cham, 2015. Springer International Publishing.

Appendix

Software and Data

You can view the project Github here. The data used to model the stocks was gathered from Wharton Research Data Services.

Contributions

Ben Kroul created the codebase. This includes the implementation of the order book, the market that interacts with the order book, the policy that acts on the market, and the market-making RL agent itself. He also helped make vital decisions about the environmental framework including which algorithms and architectures to explore.

David Guerra Nuñez contributed the TD-returns functionality, optimized the code for CUDA, and assisted extensively throughout the codebase.

Ryan Padnis spearheaded the idea for this project and contributed all finance perspectives, including selection and interpretation of the literature on high frequency limit order books and statistical estimation methods. He also ran these parameter estimation models on real stock data, as well as backtested to verify consistency, to ensure our environment was realistic. Additionally, he aided with the construction of the code base.