Mbt-gym: Reinforcement learning for model-based limit order book trading

Joseph Jerome*
Dept. of Computer Science, University of Liverpool
Liverpool, United Kingdom
j.jerome@liverpool.ac.uk

Leandro Sánchez-Betancourt*
Mathematical Institute and Oxford-Man Institute of Quantitative Finance, University of Oxford
Oxford, United Kingdom
leandro.sanchezbetancourt@maths.ox.ac.uk

Rahul Savani*
The Alan Turing Institute; and Dept. of Computer Science, University of Liverpool
Liverpool, United Kingdom
rahul.savani@liverpool.ac.uk

Martin Herdegen
Dept. of Statistics, University of Warwick
Coventry, United Kingdom
m.herdegen@warwick.ac.uk

ABSTRACT

Within the mathematical finance literature there is a rich catalogue of mathematical models for studying algorithmic trading problems such as market making and optimal execution. This paper introduces `mbt_gym`, a Python module that provides a suite of gym environments for training reinforcement learning (RL) agents to solve such model-based trading problems in limit order books. The module is set up in an extensible way to allow the combination of different aspects of different models. It supports highly efficient implementations of vectorised environments to allow faster training of RL agents. In this paper, we motivate the use of RL to solve such model-based limit order book problems, we explain the design of our gym environment, and then demonstrate its use and resulting insights from solving standard and novel problems. Finally, we lay out a roadmap for further development and use of our module for research into limit-order-book trading.

KEYWORDS

limit order book, market making, optimal execution, liquidity provision, inventory risk, reinforcement learning

ACM Reference Format:
Joseph Jerome, Leandro Sánchez-Betancourt, Rahul Savani, and Martin Herdegen. 2023. Mbt-gym: Reinforcement learning for model-based limit order book trading. In *4th ACM International Conference on AI in Finance (ICAIF '23), November 27–29, 2023, Brooklyn, NY, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3604237.3626873

*Authors contributed equally to the paper.

ICAIF '23, November 27–29, 2023, Brooklyn, NY, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0240-2/23/11.
https://doi.org/10.1145/3604237.3626873

1 INTRODUCTION

A substantial proportion of financial markets use the limit order book (LOB) mechanism to match buyers and sellers [33]. Consequently, LOBs have been a major object of study within mathematical finance. A wide range of mathematical models that capture price dynamics and order arrivals have been developed, and then trading problems, such as market making or optimal execution, have been analysed for these models. The models are differentiated primarily by the stochastic processes that drive the price and order dynamics, the actions available to the agent, and the agent's reward function.

Typically, these problems have been solved (often approximately) using standard methods from the theory of partial differential equations (PDEs), namely by formulating the Hamilton-Jacobi-Bellman (HJB) equation and using Euler schemes or finite-difference methods to solve them numerically. However: Firstly, the numerical HJB approach requires solution schemes that are tailored to the stochastic processes of the model. Ideally, solution approaches would be more *model agnostic*. Secondly, the numerical HJB approach suffers particularly strongly from the curse of dimensionality, which has also constrained the types of models that have been considered and solved this way. Thirdly, when employing the HJB approach, one often looks for semi-explicit solutions, and therefore one considers 'nice' functions as model input; this constrains the range of trading problems considered in the literature.

Reinforcement Learning (RL) is an approach to solving control problems that is based on trial and error. It has seen very rapid development since the Deep Learning revolution, with a number of prominent success stories. We contend that using RL to solve model-based LOB trading problems is important and exciting:

(i) for mathematical finance by providing a complementary solution method in addition to PDE approaches that will enable the solution of richer and more realistic models; and
(ii) for the RL community by providing a new class of problems on which to develop and understand different RL algorithms.

In relation to the highlighted weaknesses of the HJB approach, we note that by using model-free RL, one can in principle solve more complex models with assumptions that do not suit the HJB approach well, use the same learning method across many models, and solve higher-dimensional and thereby richer models. We present an open-source module that implements a range of LOB models and trading problems from the mathematical finance literature, along with model-free RL-based solution methods. Our goal is to showcase the potential of RL for solving these types of problems, provide a numerical solution to a previously unstudied problem, and facilitate further research in this exciting direction.

While RL is very promising for solving these problems, RL does have a major accepted weakness of being very sample inefficient. Fortunately, the model-based trading problems that we study allow for highly efficient implementations of vectorised environments, which we leverage in our implementations, and we find that using this vectorisation (essentially parallelisation) is crucial to get close-to-optimal solutions to the benchmark problems that we study.

Our main contributions.

- We provide an open-source repository of unified gym environments for a range of model-based LOB trading problems.
- We provide optimal baseline agents so that one can benchmark the performance of RL algorithms.
- We use a highly modular design pattern that supports combining different components of existing models, for example using a reward function from one model, with the prices and execution processes from another.
- We connect our environments to Stable Baselines 3 (SB3), which provides a suite of state-of-the-art RL algorithms. These algorithms perform robustly across a variety of tasks, allowing a "plug-and-play" approach to training RL agents on previously unstudied problems.
- We demonstrate that `mbt_gym` can quickly find a numerical solution for a popular market making problem using proximal policy optimisation [53] and provide some learning improvements which reduce the policy distance of the learnt policy from the known optimal solution.
- We introduce a novel market-making problem with self-exciting market order arrival processes and numerically find a solution, providing new insights.
- We give a roadmap for further research.

The module is available at: https://github.com/JJJerome/mbt_gym.

1.1 Literature review

1.1.1 Mathematical finance. The market making problem with limit orders was first introduced in [38] and mathematically formalised three decades later by [5]. From that point on, a large volume of research has been produced by the mathematical finance community. The work of [35] presents an explicit solution to the market making problem with inventory constraints. Examples of subsequent developments featuring more realistic models are those by [18, 36]. Other relevant extensions include the works of [13, 21] where signals are incorporated into the framework for both continuous action spaces and binary actions. The community continues to work on aspects of the market making problem, from options [7] and foreign exchange [9] to automated market making [14].[1]

Similar to the market making case, the optimal execution literature is vast; starting with [3, 11], the problem has attracted increasing attention. A significant focus of model design is on how the market processes the order flow from the liquidator (and other market participants), and how signals can be constructed and exploited [12, 15, 16, 25, 26, 31, 42, 49]. Further investigations include the case of stochastic volatility and liquidity [2], stochastic price impact [8, 27], model ambiguity [39], and latency [20]; see Donnelly [24] for a recent review. See [17, 34] for excellent textbook treatments of both optimal execution and market making.

[1]Article [14] employs `mbt_gym` for the trading problem they solve. We come back to this in Section 4.4.2.

1.1.2 Reinforcement learning for high-frequency trading literature. The following section provides a brief summary of the reinforcement learning for high-frequency trading literature, paying particular attention to the dynamics of the learning environment since that is the main focus of this paper. This section also mainly focuses on the market making problem as opposed to the optimal execution problem since the majority of the environments currently provided in `mbt_gym` focus on market making, however, a brief summary of the RL for optimal execution literature is also included.

There are broadly three main approaches to modelling the market dynamics for a reinforcement learning environment. The first is to use a "market replay" approach, in which historical data is replayed in a simulator, and the learning agent interacts with it. The second is model-based approaches, a category in which this paper sits. The final approach is that of agent-based market simulators in which hand-crafted agents that aim to model real-life market participants are allowed to interact with each other and the training agent through an exchange mechanism. A good summary of the merits of each approach is provided in Balch et al. [6].

Papers on high-frequency trading which train agents using market replay include: [41, 50], which use Nasdaq data; [54] which uses LSE data; [58] which uses CME futures data; [57], which uses XSHE data; and [30, 51, 52], which use cryptocurrency data. Some notable papers that consider market making in a model-based environment include: [22], which uses a Poisson-process-driven model similar to [32]; [44], who fit a hidden Markov model to Nasdaq trade data; [47], which considers the Poisson-process-driven model of [23]; and [55], which considers a robust adversarial version of the problem in [17, Sec. 10.2]. Finally, agent-based high-frequency trading simulators are used to train RL agents in [4, 28, 43, 45].

2 TRADING ENVIRONMENT

Our module has been designed to be extensible, general, and to minimise code duplication. For example, all our gym environments inherit from a `TradingEnvironment` class and use the same following shared `step` and "update" functions. The `step` function, shown below, is not specific to the trading problem.

```
def step(self, action: np.ndarray):
    action = self.normalise_action(action, inverse=True)
    current_state = self.model_dynamics.state.copy()
    next_state = self._update_state(action)
    dones = self._get_dones()
    rewards = self.reward_function.calculate(current_state, action,\
                                             next_state, dones[0])
    infos = self._calculate_infos(current_state, action, rewards)
    return self.normalise_observation(next_state.copy()),\
                      self.normalise_rewards(rewards), dones, infos
```

The above code, which implements the update structure shown in Figure 1, is standard in RL. Here, an action a_t and the state S_t are used to produce the next state S_{t+1}, and all three define the reward r_t that the agent receives. In the market making problem in [5], the state is composed of (i) inventory, (ii) cash, and (iii) the midprice. In the code, we distinguish further between 'agent' state – which corresponds to (i) and (ii) – and 'market' state – which corresponds to (iii) in the example. The actions in this example are the displacements from the midprice at which the agent quotes their ask and bid prices. Finally, the reward is usually a change in the value of the position adjusted by risk.

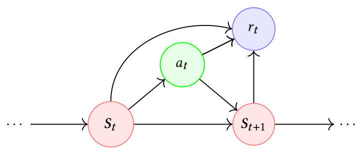

Figure 1: Directed graph representation of the step function.

The class TradingEnvironment has other model parameters that define the trading problem; e.g., terminal_time, and n_steps which are self-explanatory. It also contains the model_dynamics and the reward_function, which we describe next.

2.1 Model Dynamics

The ModelDynamics class contains the update rules for the stochastic processes used to model the financial market and the agent state. It contains the midprice_model which is common to both optimal execution problems and market making problems. It also contains arrival_model and fill_probability_model for market making problems, and price_impact_model for optimal execution problems. In addition, ModelDynamics also contains functions that are particular to the dynamics of the trading problem at hand. For example, it defines how the agent updates the state (implemented in update_state) and their action space (implemented in get_action_space).

2.1.1 *Midprice processes.*

In what follows $(W_t)_{t\geq0}$ is a standard Brownian motion. We separate the implemented midprice models into two groups: (i) midprice models without *signals* and (ii) midprice models with *signals*.

Midprice models without signals. We implement some of the most widely used models for the fundamental price process. The class BrownianMotionMidpriceModel describes the well-known arithmetic Brownian motion; here, the midprice can be written as $S_t = S_0 + \mu\, t + \sigma\, W_t$, where $\mu \in \mathbb{R}$ is the drift parameter and $\sigma \in \mathbb{R}^+$ is the volatility. Another implemented model under this category is the GeometricBrownianMotionMidpriceModel that is the well-known strong solution to the stochastic differential equation (SDE): $\mathrm{d}S_t = \mu\, S_t\, \mathrm{d}t + \sigma\, S_t\, \mathrm{d}W_t$. Lastly, we implemented BrownianMotionJumpMidpriceModel which features in [35, Section 5.2], where the midprice incorporates the impact of the filled orders, i.e., $S_t = S_0 + \sigma\, W_t - \xi^-\, M_t^- + \xi^+\, M_t^+$, where $M_t^\pm$ are the buy/sell market orders from liquidity takers and $\xi^\pm \in \mathbb{R}^+$ are the permanent price impact parameters.

Midprice models with signals. The OuMidpriceModel class accommodates models where the midprice process follows Ornstein–Uhlenbeck (OU) dynamics – examples in the algorithmic trading literature include [10, 19].

This class helps to define ShortTermOuAlphaMidpriceModel that implements popular models where the midprice process has a short-term alpha signal given by an OU process; see [16, 46, 48]. More precisely, in such models the midprice process follows

$$\mathrm{d}S_t = \alpha_t\, \mathrm{d}t + \sigma^S\, \mathrm{d}W_t^S\,, \tag{1}$$

where $\sigma^S > 0$ is the volatility of the midprice process, W^S is a Brownian motion, and $(\alpha_t)_{t\geq0}$ is an OU process satisfying

$$\mathrm{d}\alpha_t = \kappa^\alpha(\bar{\alpha} - \alpha_t)\, \mathrm{d}t + \sigma^\alpha\, \mathrm{d}W_t^\alpha. \tag{2}$$

Here, $\sigma^\alpha > 0$ is the volatility of the OU process, $\kappa^\alpha \geq 0$ is the mean-reversion speed, $\bar{\alpha}$ is the mean-reversion level, and W^α is an independent Brownian motion.

Similar to the example above, OuJumpMidpriceModel encompasses dynamics such as

$$\mathrm{d}\alpha_t = \kappa^\alpha(\bar{\alpha} - \alpha_t)\, \mathrm{d}t + \sigma^\alpha\, \mathrm{d}W_t^\alpha - \xi^-\, M_t^- + \xi^+\, M_t^+\,, \tag{3}$$

and helps define the class OuJumpDriftMidpriceModel which is the midprice (1) that features in [18].

2.1.2 *Arrivals processes.*

Poisson process. The class PoissonArrivalModel presents the most frequently used model for arrival of order flow in the market making literature. Here, the arrival of orders from market participants to buy/sell is modelled with a Poisson process $(M_t^\pm)_{t\geq0}$ with intensities $\lambda^\pm \in \mathbb{R}^+$.

Hawkes process. The self-exciting mechanism of the Hawkes processes is coded in the HawkesArrivalModel class; see Appendix A.3 in [17]. Here, for simplicity, we focus on Hawkes processes with exponential kernels. Let $(\lambda_t^\pm)_{t\geq0}$ be the stochastic intensities for buy/sell order flow and let $(M_t^\pm)_{t\geq0}$ be the associated counting processes. Then, we implement the slightly more general stochastic intensities $\mathrm{d}\lambda_t^\pm = \beta\left(\bar{\lambda} - \lambda_t^\pm\right)\mathrm{d}t + \gamma\, \mathrm{d}M_t^\pm$, where $\beta > 0$ is the mean reversion speed,[2] $\bar{\lambda} > 0$ is the baseline arrival rate, and γ is the jump size parameter.

2.1.3 *Fill probability models.*

Most models in the literature use ExponentialFillFunction as the fill probability model. Here, the probability that an order posted at depth $\delta^\pm$ is filled is given by $\mathbb{P}[\text{fill}|\delta^\pm] = e^{-\kappa\,\delta^\pm}$, for a fill exponent $\kappa > 0$; observe that if $\delta^\pm \in \mathbb{R}^+$, the resulting value lies in $[0, 1]$.[3]

Given that the fill probability model is an instance of the class StochasticProcess, this permits an exponential fill probability model with a stochastic κ that is affected by arrivals, actions, and fills. Other functional forms are straightforward generalisations. In particular, we implemented TriangularFillFunction which takes an input parameter $\delta_{\max} > 0$ (max_fill_depth in the code), and defines the probability of a fill as one if $\delta^\pm < 0$, zero if $\delta^\pm > \delta_{\max}$, and $1 - \delta^\pm/\delta_{\max}$ otherwise. This is a natural way of defining fill probabilities, but we are unaware of its use in the literature.

Similar to the previous two cases, our PowerFillFunction class, which features in [18], takes as an input a fill_exponent $\alpha >$

[2]Note that β needs to be sufficiently large for the Hawkes process to be stationary.
[3]We note in passing – and this might be a novel observation – that the HJB equation presented in Cartea et al. [17, Ch. 10.2], say, implicitly allows the fill probabilities p^+/p^- to exceed 1. This is a subtle point but relevant when solving the problem with RL methods because the strategies that correspond to fill probabilities exceeding 1 are a priori excluded.

0 and a `fill_multiplier` $\kappa > 0$, and, for $\delta^{\pm} \geq 0$ defines the probability of a fill as $1/\left(1 + \left(\kappa\, \delta^{\pm}\right)^{\alpha}\right)$.

All fill probability models have a `max_depth` property that helps to constrain the range of the action space. In particular, this value is chosen to be the value of $\delta^{\pm}$ such that the probability of not getting filled when posting at $\delta^{\pm}$ is less than 1%.

2.1.4 Price impact models. For the rest of the paper, we let 0 and $T > 0$ denote the start and end of trading, respectively, and we let $\mathfrak{T} = [0, T]$. Let $(\nu_t)_{t\in\mathfrak{T}}$ be a trading speed. When $\nu_t > 0$ the agent buys and when $\nu_t < 0$ the agent sells. Given an unaffected midprice model $(S_t)_{t\in\mathfrak{T}}$, the price impact models define how execution prices are affected by the previous/current trading activity. That is, $\hat{S}_t = S_t + I_t$, where $\hat{S}_t$ is the execution price at time t and the impact I_t is encoded in the elements of the class. The class `TemporaryPowerPriceImpact` implements a temporary (also known as instantaneous) impact of the form $I_t = k\, \nu_t^{\alpha}$, where parameter k is a `temporary_impact_coefficient` and parameter α is a `temporary_impact_exponent`.

The class `TemporaryAndPermanentPriceImpact` accommodates both temporary and permanent price impact by defining impact as

$$I_t = \underbrace{k\,\nu_t}_{\text{temporary price impact}} + \underbrace{b\int_0^t \nu_s\,\mathrm{d}s}_{\text{permanent price impact}}, \tag{4}$$

where b is the `permanent_impact_coefficient`. This is the general form of price impact one finds in Cartea et al. [17, Ch. 6].

Models where the impact is transient [31] can be implemented as long as the state remains Markovian. Certainly, our environment can accommodate transient impact with exponential kernels.

2.1.5 Implemented instances of model dynamics. The environment is equipped with various instances of `ModelDynamics`. For example, `LimitOrderModelDynamics` is used for a trader that posts limit orders similar to [5] and [17, Ch. 10.1]; `AtTheTouchModelDynamics` is for Sec. 10.2 in [15], and `LimitAndMarketOrderModelDynamics` is for [17, Sec. 8.4]; `TradinghWithSpeedModelDynamics` is for optimal execution problems such as those in Ch. 6 of [17].

2.2 Reward functions

In this section we use $(X_t)_{t\in\mathfrak{T}}$ for the cash process of the trader, $(Q_t)_{t\in\mathfrak{T}}$ for the inventory, and $(S_t)_{t\in\mathfrak{T}}$ the midprice process. The reward function `PnL` (for "profit and loss", aka risk-neutral reward) computes the changes in the mark-to-market value $Y_t := X_t + Q_t\, S_t$ of the trader's position. Specifically, the `PnL` can be written as $Y_T - Y_0$.

The reward `RunningInventoryPenalty` receives two parameters: the `per_step_inventory_aversion`, denoted by $\phi \geq 0$, and the `terminal_inventory_aversion`, denoted by $a \geq 0$. The episodic reward is then defined as the time-discretised version of

$$R_T^{\text{CJ}} = Y_T - Y_0 - a\,Q_T^2 - \phi\int_0^T Q_t^2 dt\,. \tag{5}$$

Lastly, the `ExponentialUtility` reward takes a `risk_aversion` parameter $\gamma > 0$ and computes $\exp\left(-\gamma\left(Y_T - Y_0\right)\right)$.

2.2.1 Decomposing the `RunningInventoryPenalty` reward function. When viewed from a reinforcement learning perspective, the `RunningInventoryPenalty` reward function given in (5) is undesirable due to the sparsity of the "rewards" coming from the Y_T and Q_T^2 terms (which are all received at the end of the trajectory). To improve learning, we introduce a decomposition of the episodic rewards relying on the stochastic integral representations of the terminal rewards. Explicitly, (5) can be equivalently written as

$$R_T^{\text{CJ}} = \int_0^T \mathrm{d}Y_s - a\int_0^T\left(\mathrm{d}Q_t^2 + \frac{Q_0^2}{T}\,\mathrm{d}t\right) - \phi\int_0^T Q_t^2\,\mathrm{d}t\,, \tag{6}$$

and the time-discretised version takes the form

$$R_T^{\text{CJ}} = \sum_{0\leq i<N} r(Q_{t_i}, \Delta Q_{t_i}^2, \Delta Y_{t_i}, \Delta t), \tag{7}$$

where $0 = t_0 < t_1 < \cdots \leq t_N = T$ is a lattice of points between 0 and T, $\Delta X_{t_i} = X_{t_{i+1}} - X_{t_i}$ for an arbitrary stochastic process X and

$$r(q, x, y, \Delta t) = y - a\left(x + \frac{Q_0^2}{T}\Delta t\right) - \phi q^2 \Delta t. \tag{8}$$

This incremental *reward function* is much less sparse and redistributes the terminal rewards across the episode.

2.3 Examples of currently supported models

To give a sense of the range of models that we have implemented as extensions of this base trading class, Table 1 gives examples of standard models from the literature, and Table 2 gives examples of new hybrid models that become immediately available to use by combining components from standard models. We numerically solve one of these from Table 2 using reinforcement learning in Section 4.4.

2.4 Vectorised environments

A key feature of the gym environments in `mbt_gym` is their highly parallelised nature. Single trajectories of states visited and rewards received for a given policy necessarily have very high variance. This is due to the stochasticity coming from many different places: first, there is the stochasticity of the midprice process; second, there is randomness in the arrival process; third, there is randomness in whether the agent's limit orders are filled or not.

The standard approach to parallelise reinforcement learning "rollouts" is to spin up many environments across multiple threads or CPUs and let each generate trajectories. On a single machine, this multiprocessing can be done using the `concurrent.futures` package. These trajectories are then used centrally for training.

This approach enables deep reinforcement learning algorithms to scale well with computing resources. However, the structure of model-based market making problems permits a much more efficient mode of parallelisation – namely that many trajectories can be simulated simultaneously using vector transformations from linear algebra. In particular, `mbt_gym` uses NumPy arrays [56] as the data class for states, actions and rewards. A demonstration of the speedup that this approach offers over the multiprocessing approach is shown in Figure 2. When rolling out 1000 trajectories, the multiprocessing approach takes 5 minutes and 30 seconds, whereas the NumPy approach takes 0.2 seconds.[4] In practice, when training

[4]We used an AMD Ryzen 7 3800X 8-Core Processor with 16 threads and 64GB of RAM.

Market Making Models				
Name	Arrival process	Midprice process	Model Dynamics	Reward function
Avellaneda and Stoikov [5]	`PoissonArrivalModel`	`BrownianMotionMidpriceModel`	`LimitOrderModelDynamics`	`ExponentialUtility`
Market making with limit orders [17, Section 10.2]	`PoissonArrivalModel`	`BrownianMotionMidpriceModel`	`LimitOrderModelDynamics`	`RunningInventoryPenalty`
Market making at the touch [17, Section 10.2.2]	`PoissonArrivalModel`	`BrownianMotionMidpriceModel`	`AtTheTouchModelDynamics`	`RunningInventoryPenalty`
Cartea, Jaimungal, and Ricci (CJR) [18]	`HawkesArrivalModel`	`OuJumpDriftMidpriceModel`	`LimitOrderModelDynamics`	`RunningInventoryPenalty`
Guéant, Lehalle, and Fernandez-Tapia[35] – section 5.2	`PoissonArrivalModel`	`BrownianMotionJumpMidpriceModel`	`LimitOrderModelDynamics`	`ExponentialUtility`
Optimal execution with limit and market orders [17, Section 8.4]	`PoissonArrivalModel`	`BrownianMotionMidpriceModel`	`LimitAndMarketOrderModelDynamics`	`RunningInventoryPenalty`
Optimal Execution Models				
Name	Price Impact Model	Midprice process	Model Dynamics	Reward function
Optimal Execution with Temporary and Permanent Price Impact [17, Section 6.5]	`TemporaryAndPermanentPriceImpact`	`BrownianMotionMidpriceModel`	`TradinghWithSpeedModelDynamics`	`RunningInventoryPenalty`
Optimal Execution with Transient Price Impact with exponential kernels [31]	`TransientPriceImpactExp`	`BrownianMotionMidpriceModel`	`TradinghWithSpeedModelDynamics`	`RunningInventoryPenalty`
Optimal Execution with Temporary and Transient Price Impact [49]	`TemporaryAndTransientPriceImpact`	`BrownianMotionMidpriceModel`	`TradinghWithSpeedModelDynamics`	`RunningInventoryPenalty`

Table 1: Examples of standard models from the literature implemented in `mbt_gym`

Name	Arrival process	Midprice process	ModelDynamics	Reward function
Market making with Hawkes’ order flows	`HawkesArrivalModel`	`BrownianMotionMidpriceModel`	`LimitOrderModelDynamics`	`RunningInventoryPenalty`
CJR-14 with pure jump price process	`HawkesArrivalModel`	`OuJumpDriftMidpriceModel`	`AtTheTouchModelDynamics`	`RunningInventoryPenalty`
CJR-14 with exponential utility	`HawkesArrivalModel`	`OuJumpDriftMidpriceModel`	`LimitOrderModelDynamics`	`ExponentialUtility`

Table 2: Examples of supported hybrid models

with a policy gradient approach we used 10,000 (or even 1,000,000) rollouts. This would be prohibitively slow using multiprocessing.

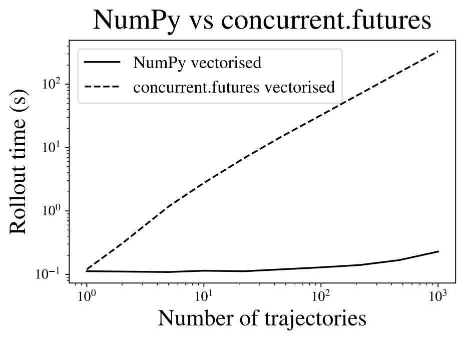

Figure 2: Speedup of vectorisation using NumPy.

3 BASELINE AGENTS

We have implemented a number of baseline agents:

(i) `RandomAgent` takes a random action in each step.
(ii) `FixedActionAgent` takes a fixed action in each step.
(iii) `FixedSpreadAgent` is a market making agent that maintains a fixed spread (symmetric around the market midprice).
(iv) `HumanAgent` allows a human to interact with the environment.
(v) `AvellanedaStoikovAgent` is optimal for [5].
(vi) `CarteaJaimungalAgent` is optimal for [17, Section 10.2].

These agents are useful for learning about models and environments. The `AvellanedaStoikovAgent` and `CarteaJaimungalAgent` are useful to benchmark solutions found by RL; in Section 4, we give an example of training an agent for the `CarteaJaimungalAgent` setting, and then compare it with the optimal baseline agent. These agents also help to test the environment; in particular, we provide two notebooks that replicate the theoretical performance of following the optimal strategies with our environment.

4 LEARNING MARKET MAKING POLICIES

In this section, we demonstrate the use of `mbt_gym` for solving market making problems. We first use it to solve a problem from the market making literature with a known closed-form solution. We highlight the resulting challenges in Section 4.2 and some approaches to overcoming them in Section 4.3. In Section 4.4, we then use `mbt_gym` to find solutions to previously unsolved problems.

4.1 Learning to solve a benchmark problem

As an example of training an agent using `mbt_gym`, we used proximal policy optimisation[5] (PPO) [53] to train an agent to solve the market making problem considered in [17, Section 10.2]. The problem considered uses: a Poisson arrival model with an arrival rate of $\lambda = 30$; arithmetic Brownian motion for the midprice process with volatility $\sigma = 0.001$ and zero drift; a terminal time of $T = 1$; and the inventory-averse reward function given in (5) with $a = 0.0001$ and $\phi = 0.0002$. This is a suitable test case for learning since an explicit optimal solution is known. We use two modifications to improve learning; the initial inventory is randomised to allow the agent to see more of the state space, and the observation and action spaces are normalised, which is known to make learning more robust [40]. Figure 3 shows the evolution of mean rewards per episode against time for different batch sizes. To roll out a batch of $n \times$ `num_steps`, we use a vectorised environment with n parallel trajectories. If the batch size is too small ($n = 10$ trajectories), then learning is unstable. If the batch size is too large ($n = 10000$ trajectories), then the (wall-clock) time until convergence starts to increase. There is little difference between $n = 1000$ and $n = 100$.

4.2 Evaluating the learnt policy

In Figures 4 and 5, we compare the learnt policy with the optimal policy from [17, Sec. 10.2]. Figure 4 shows that the RL agent manages to learn the *inventory dependence* of the policy fairly well. The agent learns to skew the bid/ask depth down/up when they hold a

[5]We used the StableBaselines3 implementation of PPO.

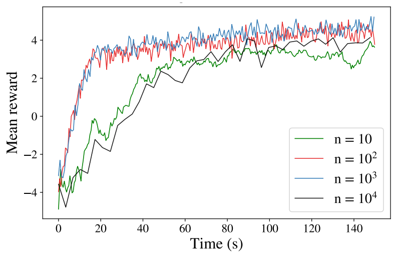

Figure 3: The effect of different batch sizes.

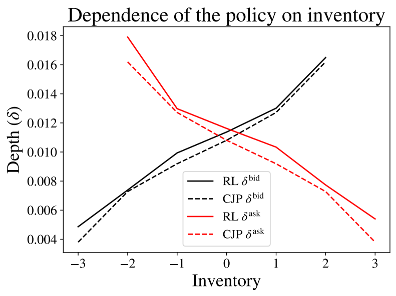

Figure 4: Trained PPO agent versus `CarteaJaimungalAgent` as a function of inventory at $t = 0.5$.

negative inventory and vice-versa. This induces mean reversion of the inventory to zero. However, Figure 5 shows a clear difference between the learnt and optimal policy as a function of time.

4.2.1 Empirically estimating the value function. We also compare the performance of the learnt RL policy depicted in Figure 4 against the closed-form optimal strategy to the continuous-time stochastic control problem in [17, Section 10.2] – both evaluated in the discrete environment on which the RL policy was trained. To do this, we perform 100,000 simulations and compute the mean performance (and standard deviation) of the cumulative rewards. The RL policy above attains 0.1933 (0.0464) and the closed-form optimal strategy to the continuous-time stochastic control problem attains 0.1954 (0.0451). Here, we performed a t-test to check if the means of the samples are different, and we found that indeed, with 99% confidence, we do not have evidence to reject the null hypothesis that the means are the same. In principle, it could even be the case that after longer training, the RL strategy outperforms the continuous-time-optimal strategy; this is because we are not evaluating the strategies in a continuous-time environment but a discrete one, where the continuous-time-optimal strategy is only approximately optimal.

4.3 Learning improvements

In this section, we explore the difference between the learnt and optimal policy as a function of time shown in Figure 5, and we discuss three improvements to learning to address this. Comparing Figures 6 and 7 to 4 and 5, it is clear that they aided policy learning.

4.3.1 Initial time randomisation. To increase the agent's exposure to certain parts of the state space, it helps to randomly sample the initial start time in addition to the initial inventory. Without this time and inventory randomisation, the agent is unlikely to see states late in the episode that have a large absolute value of inventory. Whilst this doesn't make much of a difference for learning a policy that is close to optimal in terms of rewards, we are also aiming for a policy that is interpretable and also close to optimal in the "policy space". We start more trajectories nearer the end of the episode, which we found to be better than starting at a uniformly chosen time in $[0, T]$, This is natural due to the optimal policy's dependence on time, which is greatest towards the terminal time. Our approach sampled uniformly from the interval $[0, 0.8T]$ with probability 0.5 and the interval $[0.8T, 0.99T]$ with probability 0.5.

4.3.2 Entropy regularisation [37]. This is important to ensure that we continue exploring for long enough that the fine-grained details of the policy are learnt. Without entropy regularisation, the policy can converge to a suboptimal deterministic policy too quickly. The following motivating example considers the restriction of the problem to the case $\phi = \alpha = 0$ (the agent is maximising PnL). In this case, the Bellman equation for the action-value function $q(s, \delta^+, \delta^-)$ is given by $q(s, \delta_t^+, \delta_t^-) = v(s) + \lambda^+ \delta_t^+ e^{-\kappa\delta_t^+} + \lambda^- \delta_t^- e^{-\kappa\delta_t^-}$. In particular, the optimisation that finds $\sup_{\delta_t^+, \delta_t^-} q(s, \delta_t^+, \delta_t^-)$ can be separated into two separate optimisations δ_t^+ and δ_t^-. By symmetry and since $\lambda^\pm$ is independent of $\delta^\pm$, we only consider the problem of maximising $f(\delta) = \delta e^{-\kappa\delta}$. Taking derivatives we find that the maximum is attained at $\delta = \frac{1}{\kappa}$. Suppose now that we use the standard Gaussian policy for each δ, given by $\pi(\mu_\delta, \sigma_\delta) = \mu_\delta + \sigma_\delta Z$ for parameters μ_δ and σ_δ that we are trying to learn and $Z \sim \mathcal{N}(0, 1)$. Since

$$\frac{\mathrm{d}}{\mathrm{d}\sigma}\mathbb{E}\left[f(\pi)\right] = -\frac{\mathrm{d}}{\mathrm{d}\sigma}\frac{\mathrm{d}}{\mathrm{d}\kappa}\mathbb{E}\left[e^{-\kappa(\mu+\sigma Z)}\right] = \sigma\left(\kappa\sigma^2 - \mu + \frac{2}{\kappa}\right)e^{-\kappa\mu+\frac{1}{2}\kappa^2\sigma^2},$$

if we initialise the policy with a small value for σ_δ and μ_δ far from the optimum, then will we prematurely converge to a suboptimal policy. This is because the gradient of $\mathbb{E}\left[f(\pi(\mu_\delta, \sigma_\delta))\right]$ with respect to σ_δ is always negative when $\kappa\sigma_\delta^2 < \mu_\delta - \frac{2}{\kappa}$, leading the agent to always tighten their policy and exploration to tend to zero prematurely. This effect can be clearly observed when training and extends to the non-inventory-neutral case as well.

To address this issue, we add a scaled negative entropy term to the loss. This ensures that the policy explores sufficiently throughout learning. While a large entropy term ensures that the agent will explore more throughout training, it also means that the learnt policy (whose variance will now be bounded below) is necessarily further away from the true deterministic policy. Thus, we periodically reduce the entropy scaling term during training (which is non-standard and so needed to be implemented separately).

4.3.3 Adapting the architectures of the PPO agent. Another key adjustment to improve the learning of the temporal aspect of the policy was to change the activation function for both the policy and critic in PPO from Tanh to rectified linear unit (ReLU) activations. The main benefit of using ReLU is the sparse representation it provides. This means that different parts of the policy network can learn different parts of the feature space. To allow for this sparse representation, we used larger neural networks than in Section 4.1.

4.4 Solving novel problems with RL

4.4.1 Hawkes arrivals. We now demonstrate the use of `mbt_gym` as a research tool. We train a PPO agent to solve a novel market

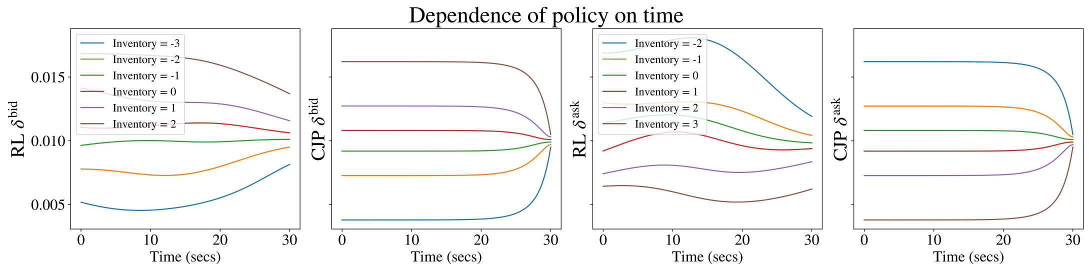

Figure 5: Learnt PPO agent versus `CarteaJaimungalAgent` as a function of time.

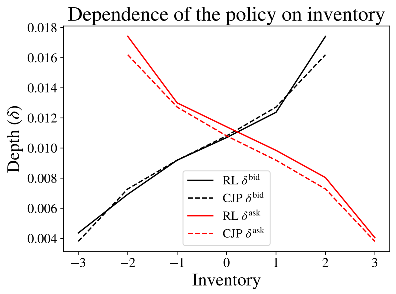

Figure 6: PPO agent with learning improvements versus `CarteaJaimungalAgent` as a function of inventory at $t = 0.5$.

making problem, a variant of the problem in Section 4.1, but where arrivals are given by a Hawkes instead of Poisson process (row 1 of Table 2). We use: a Hawkes arrival model with baseline arrival rate of $\bar{\lambda} = 10$, a jump size of $\gamma = 40$, and mean-reversion speed of $\beta = 60$; arithmetic Brownian motion for the midprice process with volatility $\sigma = 0.1$ and zero drift; a terminal time of $T = 1$; and the inventory-averse reward function given in (5) with $a = 0.001$ and $\phi = 0.5$.

Figure 8 shows that the policy learns to skew its inventory just as in Section 4.1. However, the policy now also takes the stochastic intensity $(\lambda_t)_{t\in[0,T]}$ of the Hawkes arrival process into account when quoting. In particular, when this arrival rate λ_t is higher, the agent quotes in a more inventory-neutral manner. We recall that in the risk-neutral PnL-maximisation setting, the agent's optimal quote depths are $1/\kappa$ and in our current environment $\kappa = 1$.

This should evidently be true from the HJB equation for the problem (see for example [17, equation (10.7)]). Until the end of the episode, the policy dependence on time is negligible and the time derivative of the value function drops out of the HJB equation. This means that the policy only depends on λ_t through the term $\frac{\phi}{\lambda_t}$. In particular, increasing the arrival rate has the same effect as decreasing the per-step inventory aversion ϕ – bringing the policy closer to the inventory-neutral case. From an economic standpoint, this effect occurs as the PnL term increases with λ_t, whilst the average contribution from the inventory aversion does not, meaning that the inventory aversion decreases in relative significance.

4.4.2 Automated Market Making. The recent working paper [14] used `mbt_gym` to run simulations for their novel automated market making design. Their optimisation problem can be solved in closed-form and implementing the strategy can be thought of as a market making problem where the depth of the liquidity pools are the cash and inventory of the trading agent, and the marginal exchange rate of the pool is the midprice of the traditional market making problem. In [14], the depth of the pool affects the pool's marginal exchange rate, which was implemented via a new instance of the class `ModelDynamics` and a new instance of `MidpriceModel`.

5 CONCLUSION AND FURTHER WORK

This paper serves as a proof-of-concept for the use of `mbt_gym` as a research (and didactic) tool. We showed its use to train deep RL to numerically solve a problem from the market making literature with a known solution, and then showed how it can be used to give novel insights into previously unstudied problems. In doing so, we highlighted some of the potential difficulties in using deep RL on these problems. We group possible further directions of research into two categories: algorithm research for financial deep RL and solving novel problems numerically using `mbt_gym`.

Using known solutions to families of market making and optimal execution problems [17], we can test not only the *speed of convergence* of deep RL algorithms but also the *quality* of learnt policies. This makes it an ideal testbed for novel learning algorithms. Throughout this paper, we used PPO, but an interesting extension would be to compare the accuracy of policies learnt by state-of-the-art algorithms as well as the effects of hyperparameter selection. In Section 4.3, we showed that temporal randomisation benefited optimal policy learning; an interesting research question is: can the distribution of starting states be parameterised and learnt along with the policy to accelerate learning? A related question is: how dependent is the learning speed upon the parameters of the financial market model? Answering these questions is crucial to ensure that we can trust deep RL to provide accurate numerical policies for previously unstudied problems.

Section 4.4 provided two examples where `mbt_gym` has been used to solve novel market making problems; the last two rows in Table 2 are just two examples of the limitless extensions that are possible (Section 4.4.1 was about the first row). In addition to market making, `mbt_gym` can model optimal execution problems as well.

We mention some further possible model extensions: allowing the fill probability to be influenced by order arrivals, actions, and fills themselves, e.g., in an exponential fill model, $\mathbb{P}[\text{Fill}] = e^{-\kappa\delta}$, κ could be a stochastic process that depends on the other processes;

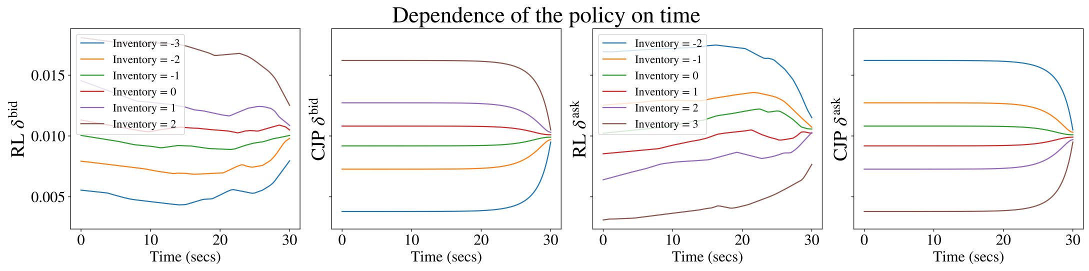

Figure 7: PPO learning agent with learning improvements versus `CarteaJaimungalAgent` as a function of time.

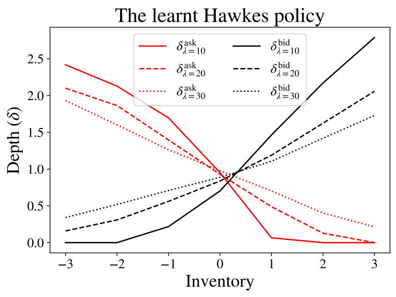

Figure 8: Inventory dependency of learnt policy for Hawkes arrival intensities $\lambda \in \{10, 20, 30\}$.

granular arrival of orders given by multi-dimensional Hawkes processes (see [1, Sec. 5]); allowing for latency (see [20, 29]).

Another interesting avenue would be to use RL to solve *families* of problems, by taking the model parameters as state features. This will improve generalisability and allow the investigation of policy dependence on parameter choice. Finally, a natural extension would be from single-agent RL to multi-agent RL, e.g., to allow the training of RL agents that are robust across model parameters by letting an adversary control the model parameter in the state as in [55].

ACKNOWLEDGMENTS

Jerome and Savani would like to acknowledge the support of a 2021 J. P. Morgan AI Research Faculty Research Award. Jerome would further like to acknowledge the support of a G-Research Early Career Grant.

REFERENCES

[1] Frédéric Abergel, Côme Huré, and Huyên Pham. 2020. Algorithmic trading in a microstructural limit order book model. *Quant. Fin.* 20, 8 (2020), 1263–1283.
[2] Robert Almgren. 2012. Optimal trading with stochastic liquidity and volatility. *SIAM J. Financial Math.* 3, 1 (2012), 163–181.
[3] Robert Almgren and Neil Chriss. 2001. Optimal execution of portfolio transactions. *Journal of Risk* 3 (2001), 5–40.
[4] Selim Amrouni, Aymeric Moulin, Jared Vann, Svitlana Vyetrenko, Tucker Balch, and Manuela Veloso. 2021. ABIDES-Gym: Gym Environments for Multi-Agent Discrete Event Simulation and Application to Financial Markets. *arXiv:2110.14771* (2021).
[5] Marco Avellaneda and Sasha Stoikov. 2008. High-frequency trading in a limit order book. *Quantitative Finance* 8, 3 (2008), 217–224.
[6] Tucker Hybinette Balch, Mahmoud Mahfouz, Joshua Lockhart, Maria Hybinette, and David Byrd. 2019. How to Evaluate Trading Strategies: Single Agent Market Replay or Multiple Agent Interactive Simulation? *arXiv:1906.12010* (2019).
[7] Bastien Baldacci, Philippe Bergault, and Olivier Guéant. 2021. Algorithmic market making for options. *Quant. Fin.* 21, 1 (2021), 85–97.
[8] Weston Barger and Matthew Lorig. 2019. Optimal liquidation under stochastic price impact. *Int. J. of Theoretical and Applied Finance* 22, 02 (2019).
[9] Alexander Barzykin, Philippe Bergault, and Olivier Guéant. 2022. Dealing with multi-currency inventory risk in FX cash markets. *arXiv preprint arXiv:2207.04100* (2022).
[10] Philippe Bergault, Fayçal Drissi, and Olivier Guéant. 2022. Multi-asset Optimal Execution and Statistical Arbitrage Strategies under Ornstein–Uhlenbeck Dynamics. *SIAM J. Financial Math.* 13, 1 (2022), 353–390.
[11] Dimitris Bertsimas and Andrew W Lo. 1998. Optimal control of execution costs. *Journal of financial markets* 1, 1 (1998), 1–50.
[12] Álvaro Cartea, Imanol Pérez Arribas, and Leandro Sánchez-Betancourt. 2022. Double-execution strategies using path signatures. *SIAM J. Financial Math.* 13, 4 (2022), 1379–1417.
[13] Álvaro Cartea, Ryan Donnelly, and Sebastian Jaimungal. 2018. Enhancing trading strategies with order book signals. *App. Math. Fin* 25, 1 (2018), 1–35.
[14] Álvaro Cartea, Fayçal Drissi, Leandro Sánchez-Betancourt, David Šiška, and Łukasz Szpruch. 2023. Automated Market Makers Designs Beyond Constant Functions. *SSRN 4459177* (2023).
[15] Álvaro Cartea and Sebastian Jaimungal. 2015. Optimal execution with limit and market orders. *Quant. Fin.* 15, 8 (2015), 1279–1291.
[16] Álvaro Cartea and Sebastian Jaimungal. 2016. Incorporating order-flow into optimal execution. *Mathematics and Financial Economics* 10, 3 (2016), 339–364.
[17] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. 2015. *Algorithmic and High-Frequency Trading.* Cambridge University Press.
[18] Álvaro Cartea, Sebastian Jaimungal, and Jason Ricci. 2014. Buy low, sell high: A high frequency trading perspective. *SIAM J. Financial Math.* 5, 1 (2014), 415–444.
[19] Álvaro Cartea, Sebastian Jaimungal, and Leandro Sánchez-Betancourt. 2023. Reinforcement Learning for Algorithmic Trading. In *Machine Learning and Data Sciences for Financial Markets: A Guide to Contemporary Practices.* Cambridge University Press.
[20] Álvaro Cartea and Leandro Sánchez-Betancourt. 2023. Optimal execution with stochastic delay. *Finance and Stochastics* 27, 1 (2023), 1–47.
[21] Álvaro Cartea and Yixuan Wang. 2020. Market making with alpha signals. *Int. J. of Theoretical and Applied Finance* 23, 03 (2020), 2050016.
[22] Nicholas Tung Chan and Christian Shelton. 2001. An Electronic Market-Maker. (2001).
[23] Rama Cont, Sasha Stoikov, and Rishi Talreja. 2010. A Stochastic Model for Order Book Dynamics. *Oper. Res.* 58, 3 (2010), 549–563.
[24] Ryan Donnelly. 2022. Optimal execution: A review. *App. Math. Fin.* 29, 3 (2022), 181–212.
[25] Ryan Donnelly and Matthew Lorig. 2020. Optimal Trading with Differing Trade Signals. *App. Math. Fin.* 27, 4 (2020), 317–344.
[26] Martin Forde, Leandro Sánchez-Betancourt, and Benjamin Smith. 2022. Optimal trade execution for Gaussian signals with power-law resilience. *Quant. Fin.* 22, 3 (2022), 585–596.
[27] Jean-Pierre Fouque, Sebastian Jaimungal, and Yuri F Saporito. 2022. Optimal trading with signals and stochastic price impact. *SIAM J. Financial Math.* 13, 3 (2022), 944–968.
[28] Sumitra Ganesh, Nelson Vadori, Mengda Xu, Hua Zheng, Prashant Reddy, and Manuela Veloso. 2019. Reinforcement Learning for Market Making in a Multi-agent Dealer Market. *arXiv:1911.05892* (2019).
[29] Xuefeng Gao and Yunhan Wang. 2020. Optimal market making in the presence of latency. *Quant. Fin.* 20, 9 (2020), 1495–1512.
[30] Bruno Gasperov and Zvonko Kostanjcar. 2021. Market Making With Signals Through Deep Reinforcement Learning. *IEEE Access* 9 (2021), 61611–61622.
[31] Jim Gatheral, Alexander Schied, and Alla Slynko. 2012. Transient linear price impact and Fredholm integral equations. *Math. Fin.* 22, 3 (2012), 445–474.
[32] Lawrence R Glosten and Paul R Milgrom. 1985. Bid, Ask and Transaction Prices in a Specialist Market with Heterogeneously Informed Traders. *J. of Fin. Econ.* 14 (1985), 71–100.

[33] Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. 2013. Limit Order Books. *Quant. Fin.* 13 (2013), 1709–1742.

[34] Olivier Gueant. 2016. *The Financial Mathematics of Market Liquidity: From Optimal Execution to Market Making*. Chapman and Hall/CRC.

[35] Olivier Guéant, Charles-Albert Lehalle, and Joaquin Fernandez-Tapia. 2013. Dealing with the Inventory Risk: A solution to the market making problem. *Mathematics and Financial Economics* 7, 4 (2013), 477–507.

[36] Fabien Guilbaud and Huyên Pham. 2013. Optimal high-frequency trading with limit and market orders. *Quant. Finance* 13, 1 (2013), 79–94.

[37] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. of ICML*. 1861–1870.

[38] Thomas Ho and Hans R Stoll. 1981. Optimal Dealer Pricing Under Transactions and Return Uncertainty. *Journal of Financial Economics* 9, 1 (1981), 47–73.

[39] Ulrich Horst, Xiaonyu Xia, and Chao Zhou. 2022. Portfolio liquidation under factor uncertainty. *The Annals of Applied Probability* 32, 1 (2022), 80–123.

[40] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. 2022. The 37 Implementation Details of Proximal Policy Optimization. In *ICLR Blog Track*.

[41] Joseph Jerome, Gregory Palmer, and Rahul Savani. 2022. Market Making with Scaled Beta Policies. *arXiv preprint arXiv:2207.03352* (2022).

[42] Jasdeep Kalsi, Terry Lyons, and Imanol Perez Arribas. 2020. Optimal execution with rough path signatures. *SIAM J. Financial Math.* 11, 2 (2020), 470–493.

[43] Michäel Karpe, Jin Fang, Zhongyao Ma, and Chen Wang. 2020. Multi-agent reinforcement learning in a realistic limit order book market simulation. In *Proc. of ICAIF*. 1–7.

[44] Adlar J Kim and Christian R Shelton. 2002. Modeling stock order flows and learning market-making from data. (2002).

[45] Pankaj Kumar. 2020. Deep reinforcement learning for market making. In *Proc. of AAMAS*. 1892–1894.

[46] Charles-Albert Lehalle and Eyal Neuman. 2019. Incorporating signals into optimal trading. *Finance and Stochastics* 23, 2 (2019), 275–311.

[47] Ye-Sheen Lim and Denise Gorse. 2018. Reinforcement Learning for High-Frequency Market Making. In *Proc. of ESANN*.

[48] Alessandro Micheli, Johannes Muhle-Karbe, and Eyal Neuman. 2021. Closed-Loop Nash Competition for Liquidity. *arXiv:2112.02961* (2021).

[49] Eyal Neuman and Moritz Voß. 2022. Optimal signal-adaptive trading with temporary and transient price impact. *SIAM J. Financial Math.* 13, 2 (2022), 551–575.

[50] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. 2006. Reinforcement learning for optimized trade execution. In *Proc. of ICML*. 673–680.

[51] Yagna Patel. 2018. Optimizing Market Making using Multi-Agent Reinforcement Learning. *arXiv:1812.10252* (2018).

[52] Jonathan Sadighian. 2019. Deep reinforcement learning in cryptocurrency market making. *arXiv:1911.08647* (2019).

[53] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv:1707.06347* (2017).

[54] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. Market Making via Reinforcement Learning. In *Proc. of AAMAS*. 434–442.

[55] Thomas Spooner and Rahul Savani. 2020. Robust Market Making via Adversarial Reinforcement Learning. In *Proc. of IJCAI*.

[56] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in science & engineering* 13, 2 (2011), 22–30.

[57] Ziyi Xu, Xue Cheng, and Yangbo He. 2022. Performance of Deep Reinforcement Learning for High Frequency Market Making on Actual Tick Data. In *Proc. of AAMAS*.

[58] Yueyang Zhong, YeeMan Bergstrom, and Amy R. Ward. 2020. Data-Driven Market-Making via Model-Free Learning. In *Proc. of IJCAI*. 4461–4468.