Skoltech

MASTER’S THESIS

Limit Order Book Queue Modelling: a Reinforcement Learning Approach

Master’s Educational Program: Data Science

Student: ____________ Bogdan Alexandrov
signature

Research Advisor: ____________ Pavel Osinenko
signature
PhD, Assistant professor

Moscow 2024

Skoltech

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Моделирование Очереди Биржевого Стакана при Помощи Обучения с Подкреплением

Магистерская образовательная программа: Науки о данных

Студент: ____________ Богдан Александров

подпись

Научный руководитель: ____________ Павел Осиненко

подпись

PhD, доцент

Москва 2024

Limit Order Book Queue Modelling: a Reinforcement Learning Approach

Bogdan Alexandrov

Submitted to the Skolkovo Institute of Science and Technology on June 20, 2024

ABSTRACT

Currently, the majority of financial trading occurs via automated trading exchanges, prompting traders and funds to utilize various trading algorithms for profit generation. These algorithms are developed, assessed, and fine-tuned through strategies such as back-testing methodologies and order book modeling, employing historical data. While extensive research has been conducted on the behavior of traders in markets, existing frameworks for parameterizing and modeling the order book often lack practical applicability. Furthermore, research predominantly relies on market-by-event data, despite market-by-level exchanges dominating trading volumes. This study seeks to bridge this gap by introducing a novel method for modeling price level dynamics within the order book using data from market-by-level exchanges and by reproducing market dynamics. The research identifies challenges and uncertainties in this process, proposing the use of reinforcement learning algorithms for resolution. The proposed model training pipeline[1] exhibits promising outcomes in mastering stochastic processes at different price levels within the order book.

Keywords: Order Book, Limit Order, Order Queue, RL

Research advisor:
Name: Pavel Osinenko
Degree, title: PhD, Assistant professor

[1]`https://github.com/BogChamp/lob-backtest`

Contents

Chapter 1
Introduction

1.1 Motivation

As of the present, the cumulative trading volume within electronic financial markets is approximated to be in the realm of several hundred trillion dollars. This substantial volume of transactions is sustained by both individual investors seeking to allocate their savings into seemingly profitable assets and institutional investors aiming to capitalize on advantageous transactions, privileged insights, and the facilitation of market liquidity. Participants engaged in the latter pursuit are commonly referred to as market makers. They harness extensive datasets to devise and validate effective trading strategies. Assessing the efficacy of these algorithms represents a pivotal yet challenging endeavor. Consequently, retrospective testing utilizing historical data, a methodology termed backtesting, is frequently employed to address this issue. Central to this practice is the restoration of the dynamics inherent within the Limit Order Book—a data repository housing records of traders' bids and asks for a given asset. The internal architecture of this structure is depicted in Figure 1.1.

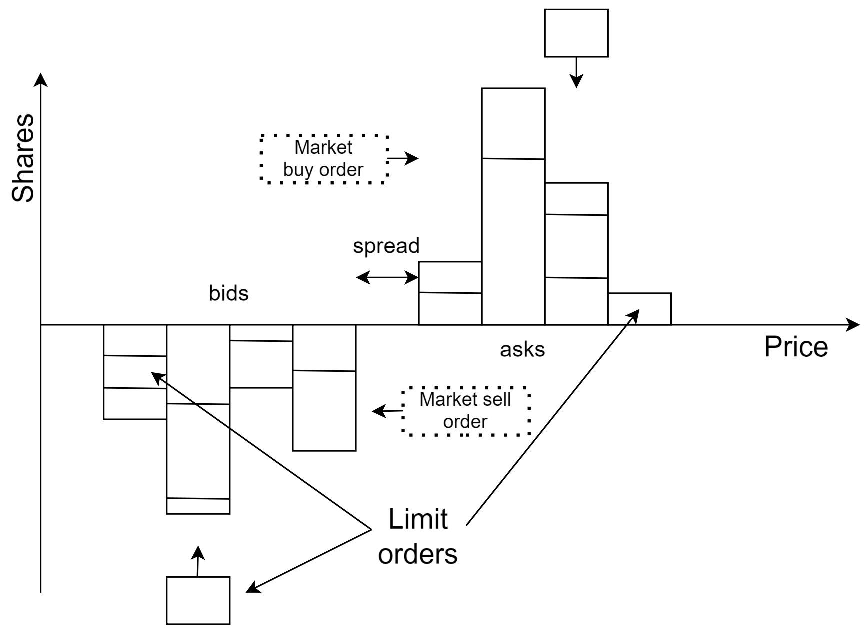


Figure 1.1: Limit order book structure.

Limit orders are categorized into two distinct types: bid, or buy orders, and ask, or sell orders. Each entry in such records includes the trader's unique identifier, the desired volume for buying or selling, and the specified trading price. Market orders, executed promptly at the prevailing optimal price for a given asset, facilitate the execution of limit orders. Consequently, limit orders contribute to the formation of a sequential price structure: buy orders are fulfilled in ascending price order, while sell orders are executed in descending order. Furthermore, in accordance with exchange protocols, a temporal sequence is observed; all limit orders sharing the same price adhere to the First In, First Out (FIFO) principle. Alongside limit and market orders, alterations and cancellations affect the dynamics of order portfolios, influencing the placement of trading volumes within the price level queue, as depicted in Figure 1.2. During the process of backtesting, it is imperative to

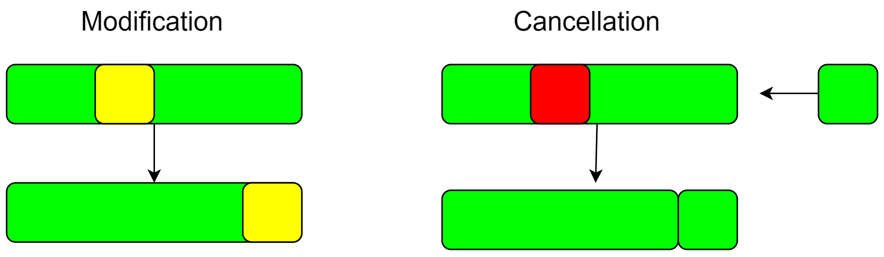


Figure 1.2: Modification and Cancellation of orders.
Green - untouched amount of asset, yellow and red are modified and cancelled respectively.

consider various factors, specifically the latency associated with the exchange, the market impact, and the order queue at the price level. The latency is contingent upon the physical integrity of the server hosting the strategy, thus underscoring the importance of scrutinizing the aforementioned points when assessing the exchange.

1.2 Existing solutions

In scholarly works [1], [2], and [4], independent Poisson processes are proposed to model the order book's dynamics, representing the distribution of limit, market, and cancel orders. Work [3] extends this model by incorporating dependent distributions and employing the Hawkes process to generate market orders. Work [6] introduces novel analytical formulas to adjust liquidity across different price levels, integrating data on order appearance and cancellation frequencies, alongside empirical observations on the exponential distribution of order lifetimes in the order book. Authors in study [7] present a comprehensive order book model, queue reactive, predicated on the interdependence of bids and asks, with request frequencies fluctuating over time, stabilizing only during intervals of unchanging asset average prices. Work [5] refines models from prior studies ([1], [2], and [4]), incorporating considerations for price jumps. Notably, order cancellations are no longer modeled by a Poisson distribution but are instead characterized by a singular parameter: the ratio of new volume to previous volume. Despite their utility in fitting real data, these studies are limited to market-by-event exchanges, where all received orders are sequentially disclosed.

1.3 Research gap

In market-by-level exchanges, traders encounter challenges due to the incomplete information provided about the order book. These exchanges furnish only aggregated volumes of orders at each price for bids and asks, periodically updated, along with details of trades, encompassing price, volume, and indications of market order purchases or sales. Analyzing the order book's dynamics using data from market-by-level exchanges introduces further uncertainty. In backtesting scenarios, if a strategy places an order at a specific price and subsequently encounters a depletion of liquidity at that price, it remains unclear whether the liquidity vanished before or after the strategy's order. Assuming a scenario where liquidity consistently precedes the strategy's order yields an optimistic evaluation of the strategy's performance, as queued orders consistently advance. Conversely, if liquidity consistently trails, the assessment becomes conservative, as queued orders occupy less favorable positions and experience fewer executions 1.3. Furthermore, the inability to observe queue dynamics and price level changes poses challenges. For instance, modifications to an order do not reflect accurately in the order book's state, nor does the departure and subsequent return of volume at a price level. Accounting for such dynamics in backtesting becomes imperative to prevent prematurely discarding potentially profitable strategies due to inadequate results.

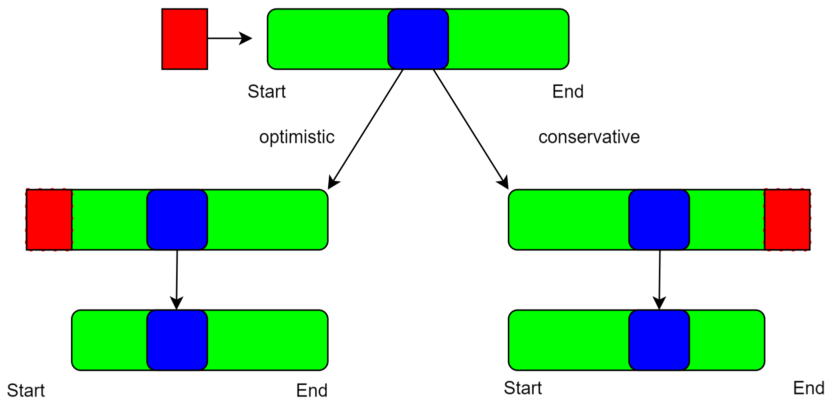


Figure 1.3: Uncertain cancellation problem.
Green - other traders liquidity. Blue - ours.

1.4 Work contribution

This study aims to contribute to the understanding of market dynamics on market-by-level exchanges, a departure from previous studies focused on market-by-event exchanges. The work introduces an algorithm designed to replicate past order book dynamics and preprocess raw exchange data for backtesting purposes. The research investigates the potential of reinforcement learning techniques to uncover latent dynamics within price levels of order books specific to market-by-level exchanges.

Chapter 2

Author contribution

- Proposed and implemented algorithm for exchange raw data preprocessing
- Conducted all the research on data
- Implemented all infrastructure, including LOB and price level dynamics, and a model train pipeline was created
- Conducted all experiments

All the work is done by the author.

Chapter 3
Literature review

One of the pioneering investigations into market microstructure is exemplified in the work [1]. This study presents theoretical frameworks aimed at modeling the order book dynamics. However, it remains primarily theoretical, lacking empirical validation with real-world market data. The proposed modeling approach is constrained, assuming a finite range of potential asset prices, and relies on the assumption of independent Poisson processes governing the distribution of limit, market, and cancelled orders. Authors in [2] build upon this foundational work by refining the model, introducing considerations such as a confined price range centered around the average price. Additionally, parameters such as the minimum price change (tick size) and a fixed order size are incorporated. Despite these enhancements, the authors acknowledge the limitations of the framework, highlighting discrepancies such as the non-Poisson distribution of cancellation orders observed empirically, the typical exponential distribution of limit orders, and the interdependence of orders submitted to the exchange. Another avenue of inquiry into market dynamics is explored in [3], where the author suggests employing the Hawkes process to analyze the order flow dynamics. Through model fitting with exchange data, the author validates the applicability of the Hawkes process in capturing market patterns and dynamics.

In the study [6] the use of an exponential distribution is employed to estimate the lifespan of limit orders, with their random size considered. This approach effectively simulates order book dynamics, as indicated by the authors' findings. These results are achieved through the introduction of novel analytical formulas that describe liquidity at various price levels, incorporating information regarding the frequency of limit, market, and cancel orders.

However, existing works in this area primarily focus on price priority rather than time priority of limit orders at each price level. They mainly investigate the temporal evolution of the order book. In contrast, the authors of [7] introduce a novel framework termed the queue-reactive model. This model not only simulates the entire order book but also assumes a bid-ask dependence. Moreover, it posits that the dynamics of limit orders are solely dependent on the current state of the order book, disregarding past events. The authors provide methodologies for evaluating market impact and order placement analysis. Additionally, they consider time priority of orders, enabling retrospective assessment of order execution likelihood. However, this model's drawback lies in its numerous assumptions regarding the frequency of limit orders and cancellations, which may not accurately reflect real-world conditions.

Another study [5] explores the price-time priority paradigm, building on previous researches [1], [2] and [4]. This extension incorporates the concept of a price jump, signifying a substantial change in an asset's average price. Furthermore, cancellation requests are no longer assumed to follow a Poisson distribution; instead, they are modeled using a continuous distribution function over the interval 0, 1, uniformly distributed across all prices. However, the primary focus of this research is on evaluating traders' positions using the theory of adverse selection. The authors demonstrate that being at the front of the queue is significantly more profitable than being at the back, underscoring the importance of realistic price level dynamics modeling during backtesting. Presently, significant research efforts are dedicated to studying order book shapes, leading to the development of various models for order book modeling.

Current scholarly literature lacks sufficient exploration into the significance of a trader's position within the price level queue, resulting in a scarcity of frameworks for its evaluation. Furthermore, existing research predominantly centers on market-by-event exchanges, which furnish comprehensive insights into the order book, thereby limiting the applicability of these models to most other market-by-level exchanges.

Chapter 4
Problem statement

4.1 Exchange data

An exploration of the backtesting procedure necessitates a comprehensive understanding of the datasets proffered by market-by-level exchanges. These are:

- Snapshot
- Trades
- Diffs

The snapshot, representing the state of the order book, offers insights into the placement of limit buy and sell orders, including the respective prices and total volume at each price level. Conversely, Trades provide a detailed record of exchange occurrences, delineating the timing, price, and quantity of transactions, along with identifying the initiator of the transaction, be it the buyer or seller. Tracking the evolution of the order book over time necessitates the utilization of diffs4.1, which serve as updates on liquidity for altered price levels. Trades are disseminated promptly upon

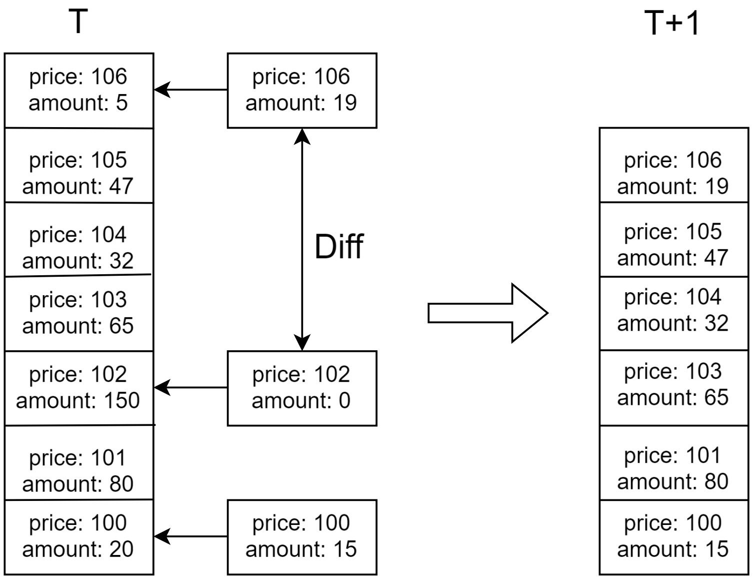
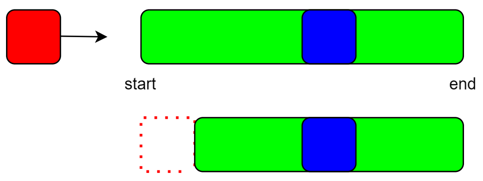


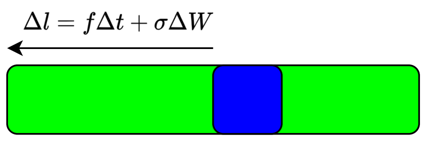
Figure 4.1: Diff updates order book

occurrence, capturing market order placements in real-time for all traders. Diffs, on the other hand, are provided at intervals dictated by the exchange, such as every 50, 100, 250 milliseconds, or varying frequencies, constituting timestamps. They encapsulate information regarding all changes in volume for every price level since the last diff. In general, trades are not indispensable for reconstructing the entire history of the order book, as diffs furnish comprehensive data for this purpose. Nevertheless, trades are essential for tracking dynamics at a finer granularity and accurately evaluating whether the strategy's limit order would have been executed in hindsight.

As previously noted, the limitations of the data make it unfeasible to discern when orders were modified or canceled, and the dynamics of price levels remain elusive. The sole observable



(a) Trade dynamic



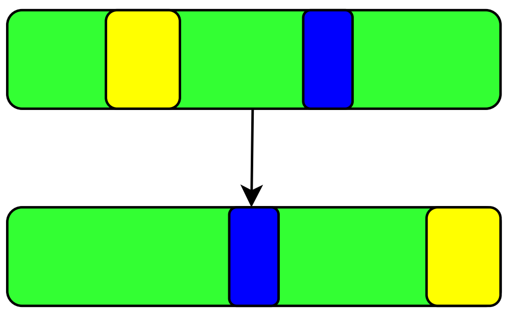
(b) Hidden queue dynamic. Here f is known impact from trades, σ - random unobservable impact from modifications and cancellations, Δl - order move to the start of queue

Figure 4.2: Queue dynamic of price level.
Green - other traders liquidity. Blue - ours. Red - amount to be executed.

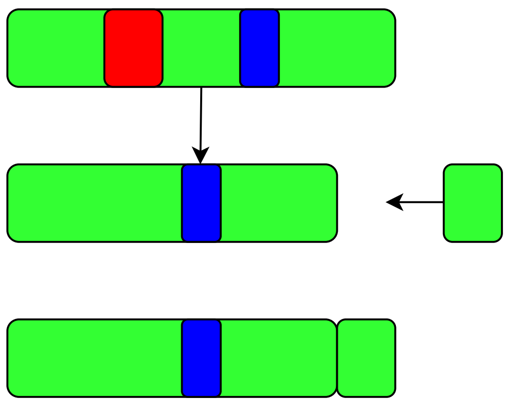
alteration in liquidity at specific price points is discernible solely through trades (see Figure 4.2), as they extract liquidity from the front of the queue upon execution.

4.2 Hidden queue dynamic

Due to the inherent limitations in accessing comprehensive order data within market-by-level exchanges, traders encounter challenges in accurately assessing the true extent of their market positions. Nevertheless, there exists an opportunity for enhancement, as illustrated in Figure 4.3.



(a) Modification of order. Yellow - amount to be modified, blue - trader's amount, green - others.



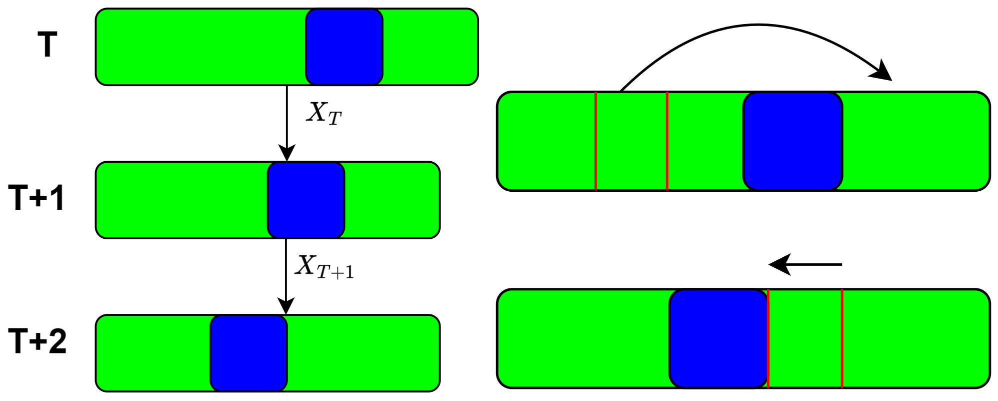
(b) Cancellation and addition of amount. Red - limit order to be canceled, blue - trader's amount, green - others. Small green frame is liquidity to be added.

Figure 4.3: Queue dynamic simulation

To model the nuanced dynamics of price level queues, it is suggested to employ reinforcement learning techniques. In this context, the primary data source derives from empirical insights garnered from operational trading strategies deployed within the stock exchange. Given the considerable expenses associated with implementing trading algorithms, it becomes necessary to artificially configure the dynamics for analytical purposes.

Definition 4.1 *Suppose we have random variable X^p with continuous distribution function $g(\bullet|O)$ on interval $[0, 1]$. Here O - some statistics of order book. Suppose random variable generate random process $\{X^p\}_t$, where t - timestamp(when diff arrived). This random process called dynamic of strategy order in the queue.*

The movement of strategy orders within the queue is obscured by a lack of direct observation. However, it is discernible that execution occurs despite the absence of explicit information from the exchange regarding our position within the price level. Notably, recent data suggests substantial liquidity preceding our position. To emulate the dynamic nature of price queues, random variables X_T are sampled from distribution g_T and applied to price levels, as illustrated below in 4.4



(a) Dynamic simulation through time

(b) Dynamic simulation per step.
Red frame is an asset amount, which moves to the end.

Figure 4.4: Queue dynamic simulation

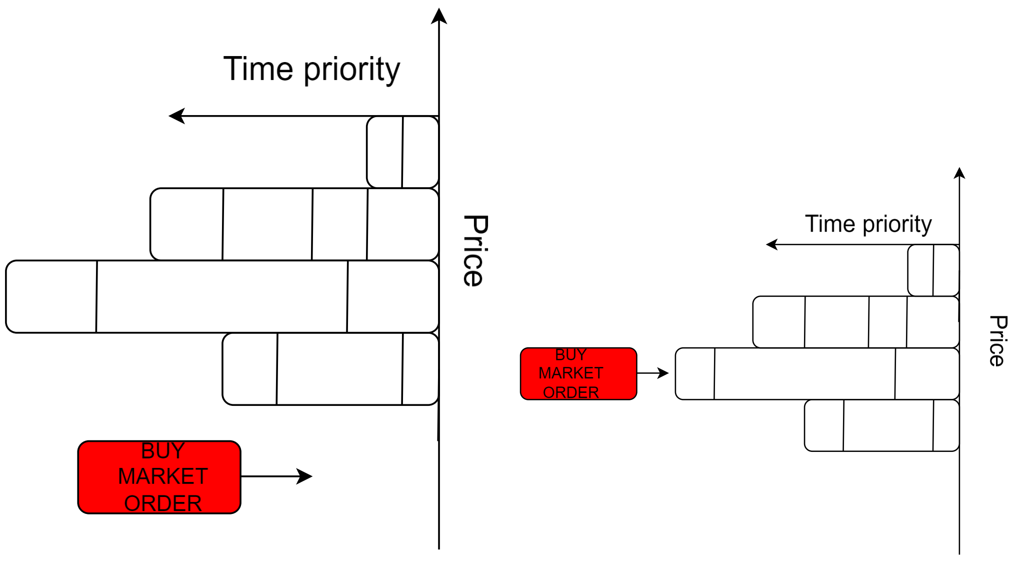
The stochastic process governing the movement of strategy orders within the queue is of paramount interest to traders and funds. Integrating this process into evaluations of trading algorithms promises greater accuracy. Consequently, incorporating this process into backtesting procedures is imperative. A proposed solution involves employing a reinforcement learning agent to learn and generate the dynamics of this process autonomously, leveraging statistical insights derived from the order book.

Chapter 5
Methodology

5.1 Data preprocessing

The initial step in conducting experiments involves preprocessing and preparing the data. As outlined earlier, exchanges furnish order book updates (diffs) and trade information, which necessitate chronological ordering to update the order book state accurately. However, not all exchanges furnish the chronological order of these updates, posing a challenge in reconstructing the order book history correctly. For this study, data from a prominent cryptocurrency exchange, Binance, was utilized, leveraging its accessible API for financial transaction data retrieval. Ethereum futures were selected as the market of interest due to their comprehensive event timing indexing.

Consequently, the sorting of all diffs and trades by time became feasible. Yet, instances arose where trade and diff times coincided, prompting the implementation of a specific procedure: if the diff contained the trade price, the trade was deemed to have occurred prior; otherwise, it was considered to have occurred subsequently. A dedicated code was developed to store data on limit orders, adhering to the order book's principles, including the prioritization of orders based on price-time precedence and state updates upon diff reception. Furthermore, a matching engine was devised to execute order logic when a market order was received. In the examination of the



(a) (b)

Figure 5.1: Discrepancies with order book actual state

order book updates (diffs) and trade data, certain disparities were identified, as delineated in Figure 5.1. There were instances where the exchange reported trade information at a price point where no corresponding liquidity was present in the order book. This anomaly suggests the emergence of limit orders at that particular level between the occurrence of the trade and the last received diff. Conversely, there were occurrences where trades transpired at prices inferior to the best observable offer in the market, indicating the cancellation of limit orders between the trade and the diff update. These discrepancies stem from the exchange's intermittent provision of order book updates, occurring at specific timestamps rather than continuously. Given the substantial volume of market events unfolding every second, the discretization of diffs impedes a precise observation of the true market state. Therefore, to ensure coherence between the processing of trades and the current state of the order book, the identified latent market dynamics were incorporated into the event sequence alongside diffs and trades. Timestamps were managed as follows: if a hidden event could plausibly have transpired at diff timestamp, it was attributed to that diff. Otherwise, the earliest feasible time for the hidden dynamics to have occurred without compromising the consistency of the order book and trades was determined.

5.2 Environment

The system in our setup is an order book, in which a strategy's limit order exists. This limit order must be managed by an RL agent, pushing it forward in the queue at the price level. The environment for the agent was designed as follows:

- The state of the system is the current order book entry with all historical dynamics.
- As an observation, a tuple of three numbers was taken: how many updates to the order book the order lived, changes in the volume of the price level on which the order is located, and the average price of the asset.
- The action is a number from 0 to 1, which indicates which part of the volume ahead of our order should move behind.
- The step of the simulation is the update of the order of the book by the diff.
- One episode begins with the setting of a limit order and ends with its cancellation or execution.
- The sum of rewards per episode is modulo 1. If, when modeling hidden dynamics, the order was executed at the same time as in reality, then the episode reward is 1; otherwise, it is -1. Next, it is evenly distributed among all the steps.

5.3 Ground truth data

In the pursuit of generating realistic dynamics within the order queue at various price levels, it is proposed to employ a stochastic process. This entails the utilization of a random number generator ranging from zero to one. Subsequently, the development of an algorithm for order placement within the order book becomes imperative. However, employing entirely random price levels proves unsuitable for the training of a reinforcement learning (RL) agent. Placing orders at price levels where no executions have occurred renders the receipt of rewards impossible, thereby hindering the assessment of the accuracy of queue dynamics modeling. Similarly, choosing price levels entirely executed by a single market order proves suboptimal, as the simulated dynamics become inconsequential, with limit orders executed irrespective of their queue position. To address this challenge, historical data analysis is conducted to identify price levels influenced by trades yet not entirely executed by a single order. As a strategic approach, an algorithm is devised to place orders on these identified price levels coinciding with their appearance in the order book, positioning them at the queue's tail. Additionally, an auxiliary tactic is implemented during placement, aimed at:

- All orders are one asset tick in size. Since we only care about the fact of whether they will be executed or not, the size does not matter. Moreover, such a choice immediately removes the possibility of partial execution when only part of the order has been fulfilled.

- The volume of positions placed does not increase the liquidity of the order book. Placing an order means assigning historical volume to our strategy.

In the context of backtesting, a scenario may arise wherein the strategy opts to establish a position at a price level where its limit order already resides. In such instances, this liquidity is superseded, thereby not contributing to loss formation. Furthermore, unexecuted positions are disregarded, given that their actual positioning during strategy implementation remains uncertain.

5.4 Backtest

The backtest process with simulation of queue dynamics consists of several sequential steps: updating the order book using a diff, sending limit orders according to a strategy, applying trades, and applying queue dynamics.

Algorithm 1: Backtest loop

```
   Data: Diffs, trades, suitable price levels
   Input: Action generator, or agent.
1  Sample timestamps $t_i$ and price levels $p_i$, at which strategy will place orders. Name it X.
2  t = 0
3  while $t < T$ do
4      Place strategy's order from X, which should be set at timestamp t. Indicate if
         overwriting has occurred.
5      Apply trades to the order book, which occurred between timestamps $t$ and $t + 1$. If
         the strategy's position was executed, store information about it.
6      Sample actions. Move the set orders forward in the queue.
7      Apply diff, which arrived at timestamp t.
8      t = t+1
```

Table above shows the pipeline of the backtest procedure. It requires that all diffs and trades be sorted in ascending order by time. T is a final state when the simulation is over. This pipeline is suitable both for simulating real positions and for training an agent.

5.5 RL algorithms

In our case, both observations and actions have a continuous space. Therefore, RL algorithms from the policy-based family are suitable for this environment. The following algorithms were considered: REINFORCE, Actor-Critic and PPO.

REINFORCE

This method consists of learning stochastic policy using gradient methods. The formula for updating the model weights is shown below 5.1.

$$\theta_{i+1} = \theta_i + \alpha_i \frac{1}{M} \sum_{j=1}^{M} \Big(\sum_{k=0}^{N_j-1} \sum_{l=k}^{N_j-1} \gamma^l r(y_l^j, u_l^j) \nabla_\theta \ln \rho^\theta (u_k^j \mid y_k^j) \big|_{\theta=\theta^i} \Big) \tag{5.1}$$

where θ is neural network parameter, α - learning rate, γ - discount rate, M - number of episode, N_j - number of steps of each episode, r - reward or cost function, ρ - action distribution, u - action sampled, y - observation.

In the context of reinforcement learning, the utilization of a normal distribution sampler is a common strategy for policy determination, where the mean is forecasted through a neural network. The sampled outputs represent actions directed towards controlled entities. However, a notable drawback of the REINFORCE algorithm lies in its susceptibility to substantial variance in gradient approximations, which may result in prolonged convergence periods and training instability. In response to this challenge, the REINFORCE algorithm is often augmented with a baseline mechanism to mitigate these issues:

$$\theta_{i+1} = \theta_i + \alpha_i \frac{1}{M} \sum_{j=1}^{M} \Big(\sum_{k=0}^{N_j-1} \Big(\sum_{l=k}^{N_j-1} \gamma^l r(y_l^j, u_l^j) - B_k \Big) \nabla_\theta \ln \rho^\theta(u_k^j \mid y_k^j) \big|_{\theta=\theta^i} \Big) \tag{5.2}$$

where $B_k = \frac{1}{M}\sum_{j=1}^{M}\sum_{k'=k}^{N-1}\gamma^{k'} r(y_{k'}^j, u_{k'}^j)$ is baseline, calculated from previous iteration. Initial values are zero.

Actor Critic

The Actor-Critic algorithm merges elements of policy-based methods (Actor) and value-based methods (Critic) in reinforcement learning. This fusion aims to overcome the individual limitations of each approach. The Actor learns the policy, and the update rule for it is shown in the formula 5.3:

$$\theta_{i+1} = \theta_i + \alpha_i \frac{1}{M} \sum_{j=1}^{M} \sum_{k=0}^{N_j-2} \gamma^k \left(r(y_k^j, u_k^j) + \gamma \hat{J}^w(y_{k+1}^j) - \hat{J}^w(y_k^j) \right) \nabla_\theta \ln \rho^\theta(u_k^j \mid y_k^j) \big|_{\theta=\theta^i} \tag{5.3}$$

Here $J(y_k)$ is a value function of observation y_k, evaluated by the critic. In other words, the Critic evaluates the average reward that can be received while in a given state. But the Critic also needs training. Temporal Difference Learning is used for this. The essence of this method is to minimize the so-called TD error, which is equal to the advantage function: $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$. Here $V(s_t)$ is a value function estimated by Critic for the state s_t. $Q(s_t, a_t)$ is a q-value, which is estimated with a TD target, or $Q(s_t, a_t) = R_t + \gamma R_{t+1} + \ldots + \gamma^{N_{TD}} V(s_{t+N_TD})$. Here N_{TD} is the length of temporal difference learning or, in other words, how many steps are used to evaluate the q-function. However, the Critic needs to be updated with each episode. The learning algorithm is shown in the table below.

```
Algorithm 2: Critic training
  Input: $N_{TD}, N_{epoch}$
1 for $e = 1$ to $N_{epoch}$ do
2     for $j = 1$ to $M$ do
3         $w^{new} = w^{old} - \eta \nabla_w \left[ \frac{\sum_{k=0}^{N_j-1-N_{\mathrm{TD}}} (\hat{J}^w(y_k^j) - r(y_k^j, u_k^j) - \gamma r(y_{k+1}^j, u_{k+1}^j) - \ldots - \gamma^{N_{\mathrm{TD}}} \hat{J}^w(y_{k+N_{\mathrm{TD}}}^j))^2}{N_j-1-N_{\mathrm{TD}}} \right] \Big|_{w=w^{old}}$
```

Thus, in order to train the Actor-Critic method, two additional hyperparameters must be set:

how many epochs train critic per iteration and the length of TD.

Proximal Policy Optimization

It is a popular reinforcement learning algorithm that aims to improve traditional policy gradient methods such as REINFORCEMENT by eliminating some of their limitations, such as high variance in learning. PPO achieves this by limiting the size of the policy update at each iteration, which helps to stabilize training and prevent significant policy changes that can lead to lower performance. This makes PPO more efficient and reliable compared to other policy gradient methods. This effect is achieved by fixing the maximum difference between the distributions of the old and new policies.

Algorithm 3: PPO agent training

```
   Input: $N_{epoch}, \epsilon$
1  for $e = 1$ to $N_{epoch}$ do
2      $\theta^{\text{new}} = \theta^{\text{old}} - \alpha \nabla_\theta \left( \frac{1}{M} \sum_{j=1}^{M} \sum_{k=0}^{N_j - 2} \gamma^k \max \left( \hat{A}^w(y_k^j, u_k^j) \frac{\rho^\theta(u_k^j | y_k^j)}{\rho^{\theta_i}(u_k^j | y_k^j)}, \hat{A}^w(y_k^j, u_k^j) \, \text{clip}_{1-\varepsilon}^{1+\varepsilon} \left( \frac{\rho^\theta(u_k^j | y_k^j)}{\rho^{\theta_i}(u_k^j | y_k^j)} \right) \right) \right) \Big|_{\theta = \theta^{\text{old}}}$
```

New hyperparameters reappear here: epochs for agent training and ϵ, the maximum difference in the ratio of the probabilities of actions for the old and new policies. Also, here $\hat{A}^w(y_k^j, u_k^j) = r(y_k^j, u_k^j) + \gamma \hat{J}^w(y_{k+1}^j) - \hat{J}^w(y_k^j)$ and $\theta_{old} = \theta_i$, $\theta_{new} = \theta_{i+1}$. With this scheme, we only ignore the change in probability ratio when it would make the objective improve, and we include it when it makes the objective worse.

5.6 Neural Networks

The policy model was taken as

$$\rho^\theta(u \mid y) = \mathsf{pdf}_{\mathcal{N}\left(\lambda \mu^\theta(y) + \beta, \lambda^2 \sigma^2\right)}(u) = \mathsf{pdf}_{\mathcal{N}\left(\mu^\theta(y), \sigma^2\right)} \left(\frac{u - \beta}{\lambda} \right) \tag{5.4}$$

where $\mathsf{pdf}_{\mathcal{N}(\bullet_1, \bullet_2)}$ refers to the normal probability density with mean $\bullet_1$ and (co)variance $\bullet_2$, $\beta = \frac{u_{\text{min}} + u_{\text{max}}}{2}$, $\lambda = \frac{u_{\text{max}} - u_{\text{min}}}{2}$. u_{min} and u_{max} are here the boundaries of the value for the action. In our case, they are 0 and 1, respectively. The $\mu^\theta(\cdot)$ is a perceptron with weights θ:

$$\mu^\theta(y) : y \rightarrow \text{Linear(3, ...)} \rightarrow \text{LeakyReLU(0.2)} \rightarrow \text{...} \rightarrow \text{Linear(..., ...)} \rightarrow (1 - 3\sigma) \tanh \left(\frac{\cdot}{L} \right) \tag{5.5}$$

In our case, the perceptron should not be too large; otherwise, the backtest will take a huge amount of time. And in this case, it will be impossible to evaluate strategies since there may be many variations of them, and the simulation will take several hours due to the large amount of data. σ here is a preset standard deviation, and L is a hyperparameter, which helps in the convergence of a model.

The model for the Critic is also an ordinary linear perceptron.

Chapter 6
Numerical experiments

The following dynamics were selected for the experiments:

- ”Zero” dynamic. In this case, there is no dynamics as such. This scenario potentially mirrors conditions found within a highly liquid market characterized by a significant asset tick. Empirically, it is noted that at the optimal price levels, there exists a reduced frequency of order cancellations and modifications. This phenomenon can be attributed to traders’ inclination towards prompt fulfillment.
- ”Full forward” dynamic. This mechanism prioritizes all submitted orders at the forefront of the queue. This scenario reflects traders with minimum latency between their orders and the exchange. Consequently, their orders gain expedited entry into the order book, positioning them ahead of others. Alternatively, this phenomenon may indicate a volatile market environment characterized by frequent order cancellations.
- A generator for the uniform distribution of numbers from 0 to 0.1. In this case, the orders will move in the queue a little bit each step of the simulation.

The order book works in a ”pessimistic” mode, bearing in mind that the outflow of liquidity from the diffs goes first behind our orders and only then in front.

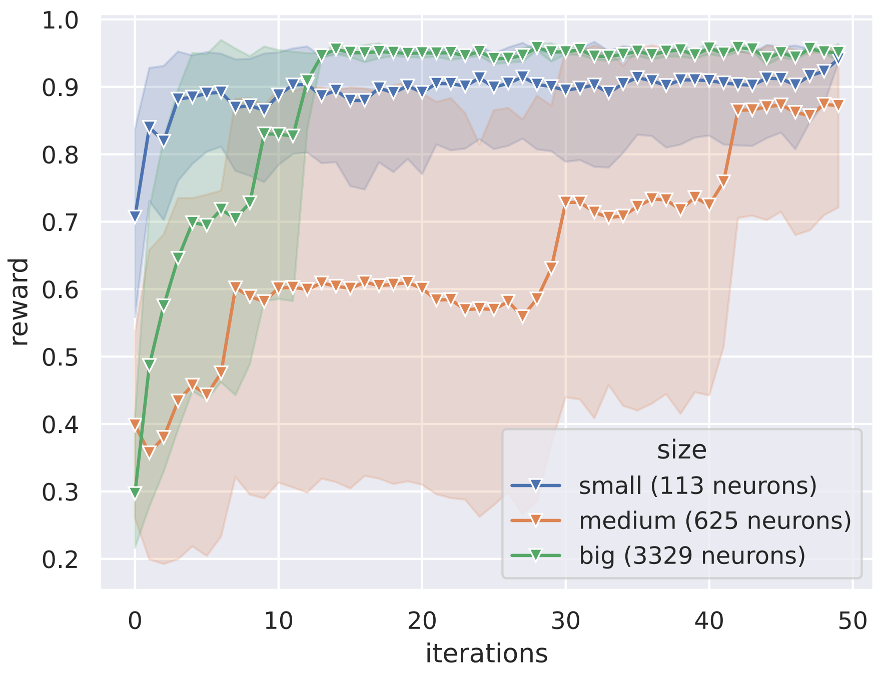


Figure 6.1: The effect of the agent’s model size on the reward

To begin with, it was decided to consider what the size of the agent model should be. For this purpose, we took three neural networks of different sizes and trained them on ”zero” dynamic using the REINFORCE method. The results are shown in the figure 6.1.

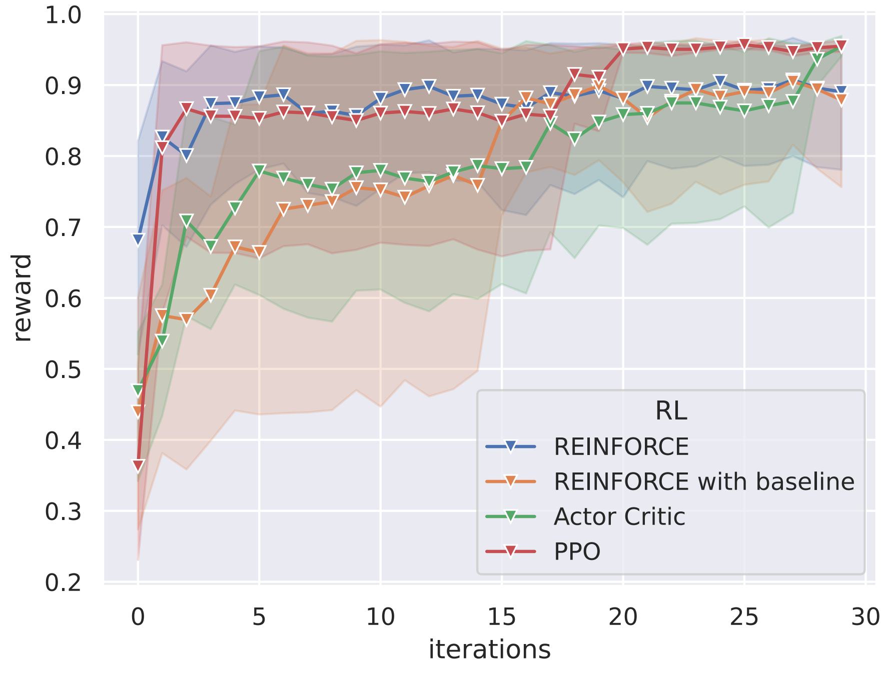


Figure 6.2: ”Zero” dynamic reward curves

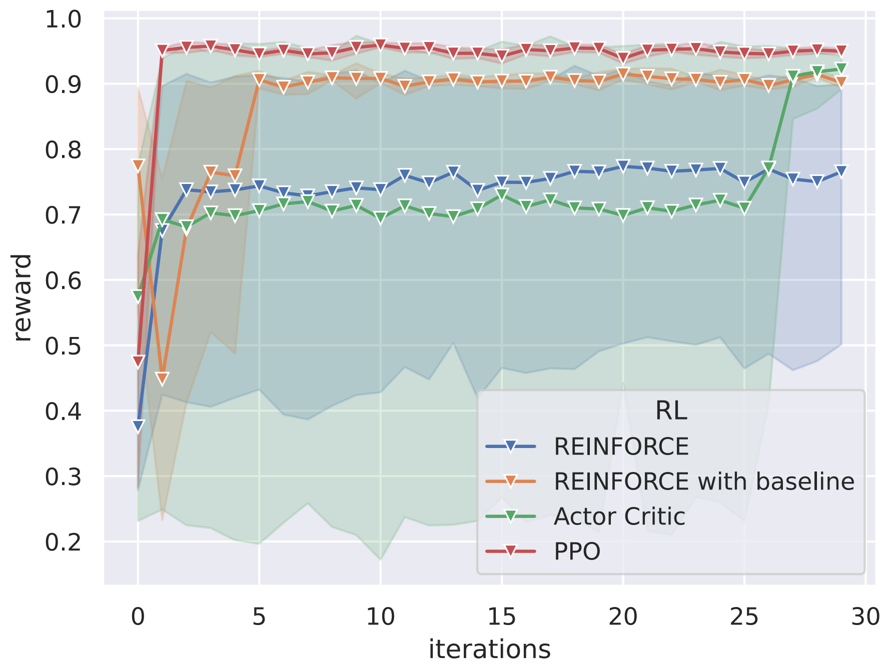


Figure 6.3: ”Full forward” dynamic reward curves

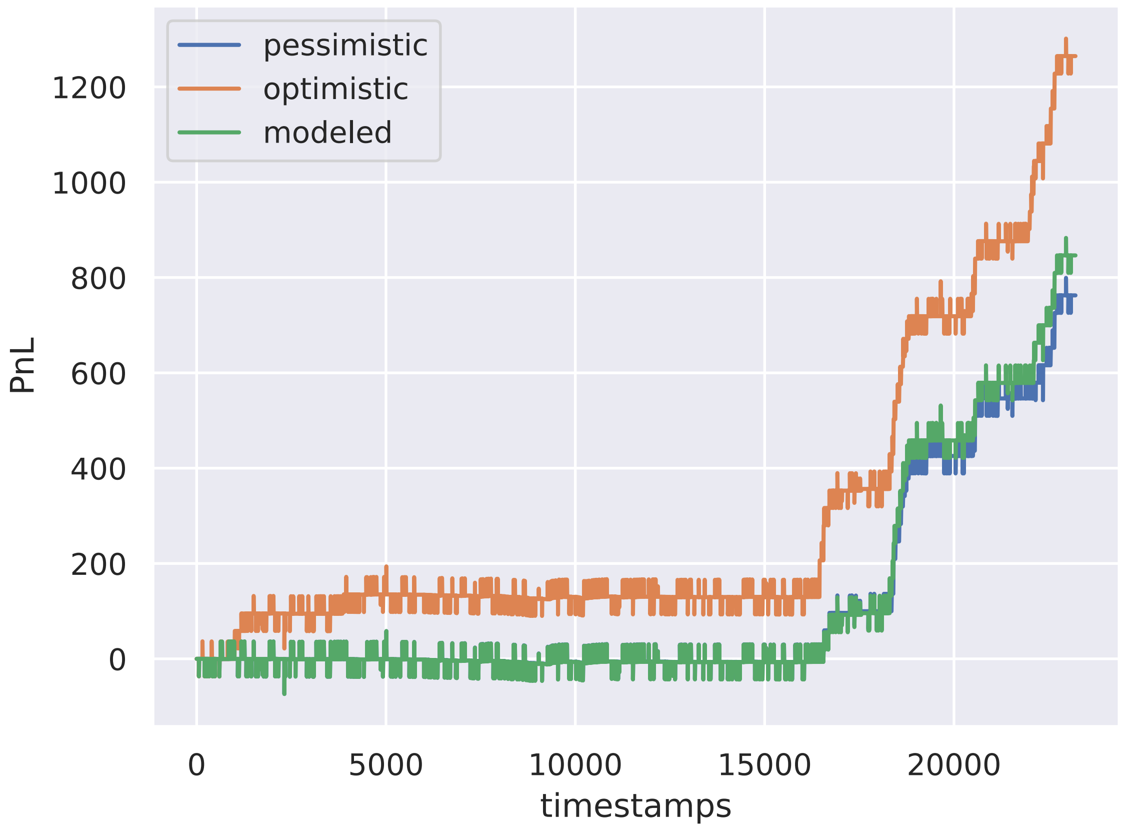


Figure 6.4: Hidden dynamics affect the evaluation of a trading strategy. The graph shows that by modeling this unobservable process, you can get a different assessment of the trading algorithm.

Chapter 7
Discussion and conclusion

To model the dynamics of a price level queue, certain assumptions were made regarding its characteristics, such as its stability or volatility. Reinforcement learning agents successfully learned the proposed queue dynamics. Notably, the Proximal Policy Optimization (PPO) method demonstrated superior performance, boasting the highest convergence rate and optimal reward outcomes. However, other methods exhibited considerable variance in reward convergence. It is noteworthy that even modestly sized neural networks proved capable of grasping these dynamics, as evidenced by comparisons between rewards and network sizes. Reinforcement learning techniques exhibit promising results in instructing the stochastic process within the order book queue.

As a prospective avenue for further investigation, it would be of scholarly interest to implement the proposed pipeline in real-time within an established exchange framework. This endeavor would facilitate the training of the agent using authentic data, potentially yielding novel insights into the operational dynamics of the exchange.

Acknowledgements

Many thanks to my supervisor Pavel Osinenko for mentoring, PhD student Georgy Malania for ideas and curation, as well as the AIDA laboratory.

Innovations

In my master thesis, I have pioneered an innovative approach to enhance the quality of limit order book backtesting by leveraging reinforcement learning. Traditional methods often struggle with modeling the complex interactions and dynamics within the First-In-First-Out (FIFO) queue of order books. My approach addresses these limitations by employing reinforcement learning algorithms to accurately simulate the behavior and progression of orders within the queue. By modeling the dynamic placement and execution of orders, my method provides a more realistic and precise backtesting environment. This advancement not only improves the fidelity of backtesting results but also offers deeper insights into the strategic impact of order placement and execution strategies, paving the way for more informed decision-making in algorithmic trading.

Bibliography

[1] Domowitz, I., and Wang, J. Auctions as algorithms. *Journal of Economic Dynamics and Control 18* (1994).

[2] Eric Smith, J. Doyne Farmer, L. G., and Krishnamurthy, S. Statistical theory of the continuous double auction.

[3] Hewlett, P. Clustering of order arrivals, price impact and trade path optimisation.

[4] J. Doyne Farmer, P. P., and Zovko, I. I. The predictive power of zero intelligence in financial markets. *Proceedings of the National Academy of Sciences of the United States of America* (2003).

[5] Moallemi, C. C., and Yuan, K. A model for queue position valuation in a limit order book. *SSRN Electronic Journal* (2016).

[6] Toke, I. M. The order book as a queueing system: average depth and influence of the size of limit orders. *Quantitative Finance 15* (2013).

[7] Weibing Huang, C.-A. L., and Rosenbaum, M. Simulating and analyzing order book data: The queue-reactive model. *Journal of the American Statistical Association 110* (2013).

Appendix

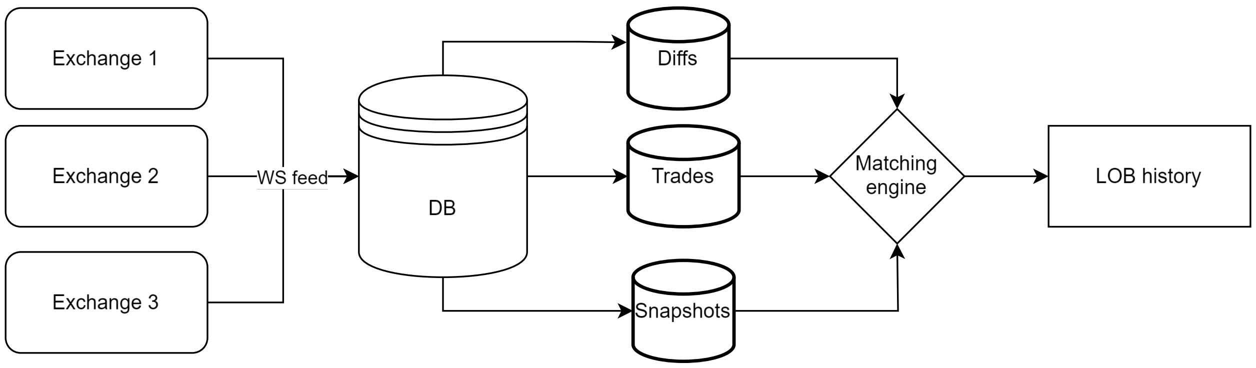


Figure 7.1: Data pipeline for backtest.

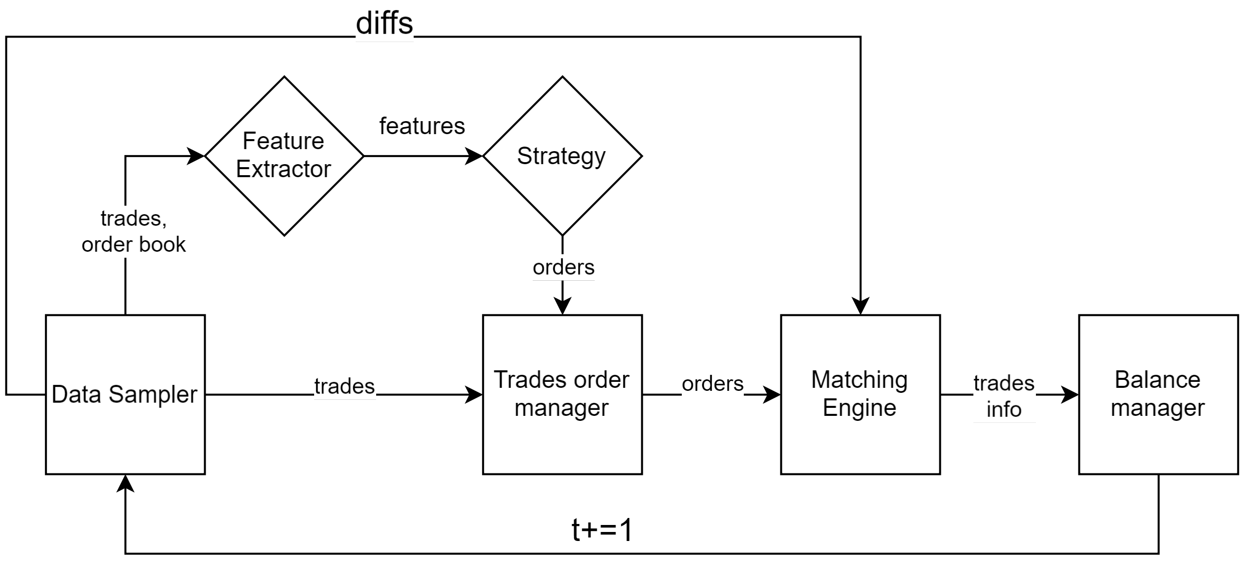


Figure 7.2: Backtest loop.

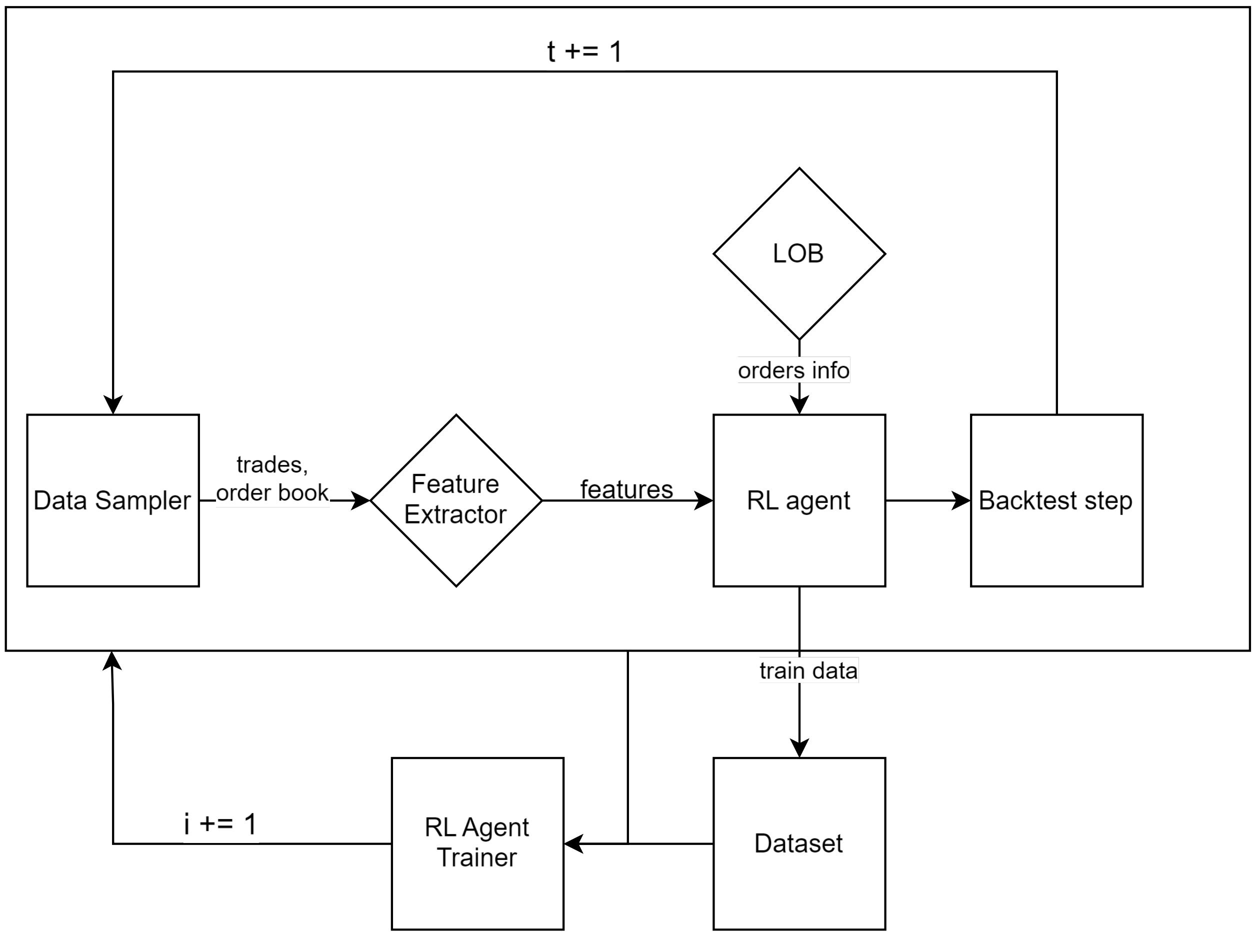


Figure 7.3: Reinforcement learning agent training scheme.
t - timestamp, i - iteration of training.