Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Exploring Trading on Decentralized Exchanges using Reinforcement Learning

Master's Thesis

Jaye Danéa Plüss

jpluess@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Ye Wang, Benjamin Estermann
Prof. Dr. Roger Wattenhofer

May 4, 2023

Acknowledgements

I would like to extend my genuine gratitude to those who have offered invaluable assistance and support throughout the course of my thesis completion. Firstly, I would like to thank Ye Wang and Benjamin Estermann for their consistent guidance and support in the areas of decentralized finance and machine learning implementation, respectively. Their expertise and insight have been instrumental in shaping my research and its final outcome. I would also like to thank Prof. Dr. Wattenhofer for his guidance and redirection of my thesis at the midterm, which helped me to refine my research and achieve better outcomes. Additionally, I would like to acknowledge the contributions of the present Ph.D. candidates at the midterm, whose valuable input and feedback have helped me to improve my work. Their suggestions and comments have been instrumental in shaping the final outcome of my research.

Abstract

In this thesis, we investigated the use of reinforcement learning for trading on decentralized exchanges, with a particular focus on the Uniswap V2 platform. To facilitate this exploration, a simulation was developed to replicate the transaction dynamics of such an exchange. Experiments were initially conducted using artificial blockchain data, followed by the incorporation of real historical data.

A multi-stage learning pipeline was designed to optimize performance, addressing the challenges related to hyperparameter tuning in reinforcement learning and offering insights into effective and less effective approaches. Experiments incorporated both short-term and long-term rewards, with the latter integrating customized preferences into the agent's policy concerning liquidity provision and risk readiness.

Furthermore, a method for handling action spaces requiring both continuous and discrete values was proposed by encoding the discrete actions. Two approaches for dealing with variable-sized observation spaces were also presented, employing either a padding layer or an attention layer as the network's initial layer.

The experiments involved a stable coin pair (USDC-USDT) and an unstable coin pair (ETH-USDC), highlighting the distinct approaches associated with each dataset. The performance achieved with the stable coin pair was linked to a trading strategy that combined cryptocurrency swapping and liquidity provision. The unstable coin setting posed greater complexity, and insights were provided into potential difficulties when training an RL agent for trading in such a setting.

Overall, the findings illustrate the potential of reinforcement learning in cryptocurrency trading, attaining positive results for both stable and unstable coin pairs.

Contents

Chapter 1

Introduction

Decentralized Finance (DeFi) has emerged as a powerful force in the financial industry, providing an alternative to centralized financial systems by offering trustless, permissionless, and decentralized access to financial services. DeFi protocols are built on blockchain networks and are characterized by their transparency, security, and accessibility. These protocols enable anyone with an internet connection to participate in various financial activities, such as lending, borrowing, and trading, without the need for intermediaries like banks and financial institutions. Uniswap [1], one of the leading DeFi protocols, has seen massive growth in the past year, reaching over 2 billion USD in daily trading volume [2]. Uniswap is a decentralized exchange that operates on the Ethereum blockchain and enables users to trade cryptocurrencies in a permissionless and non-custodial manner.

Machine learning techniques have demonstrated a substential potential in enhancing the performance of trading strategies in the financial domain [3, 4]. Specifically, reinforcement learning (RL) is being researched to achieve this goal [5]. By allowing an agent to learn decision-making based on rewards and penalties received from its environment, RL is ideal for developing intelligent trading strategies that can adapt to fluctuating market conditions and maximize returns.

In this thesis, the primary objective is to investigate the potential of RL in enhancing the performance of trading strategies. The research will focus on Uniswap, which is one of the most widely used decentralized exchanges (DEX) in the market. The first step will be to build a simulation of the Uniswap liquidity pool that will accurately reflect its behavior under different market conditions. On top of the simulation, a reinforcement learning pipeline will be developed to interact with the simulation, commonly referred to as the environment, with the goal to learn how to optimize trading strategies.

Initially, experiments will be conducted using artificial blockchain data to assess the implementation's behavior and identify the potential benefits of various approaches. After ensuring a stable implementation, a real dataset based on historical data will be generated to evaluate the strategies' performance in a more realistic context. The agent will then be trained on this real simulation to develop strategies that have the potential to perform well in real-world scenar-

ios. The performance of the developed strategies will be compared with those of traditional market-making and arbitrage strategies. The ultimate goal is to identify new trading strategies that can outperform traditional methods in terms of returns, liquidity provision, and risk management.

Chapter 2

Background and Related Work

2.1 Decentralized Finance

Decentralized finance, commonly known as DeFi, is a rapidly expanding movement that aims to create a more open, accessible, and transparent financial system. This shift was enabled through the introduction of smart contracts on the Ethereum blockchain [6]. A smart contract is a self-executing program that contains a set of pre-defined conditions and instructions governing the execution of the contract. These conditions and instructions are encoded in the smart contract code and stored on the blockchain. When certain conditions are met, the smart contract is triggered, and the instructions are executed automatically. For example, a smart contract for a peer-to-peer lending platform may have conditions that specify the amount of the loan, the rate of interest, and the repayment conditions. Once these conditions are met, the smart contract automatically transfers the funds to the borrower and enforces the repayment schedule [7, 8]. Ethereum's programming language, Solidity [9], allows developers to create custom smart contracts that can be used in a variety of DeFi applications such as decentralized exchanges, lending platforms, and insurance protocols. The underlying blockchain technology ensures the transparency and immutability [10] of the transactions, while smart contracts automate the execution of the agreements between parties.

2.2 Decentralized Exchanges

Decentralized exchanges, commonly referred to as DEXs, have become increasingly popular in recent years due to their ability to offer trustless, transparent, and secure transactions. Traders sell and buy assets on DEXes by interacting with smart contracts [7] on the blockchain without the involvement of centralized authorities. One of the most established DEXs is Uniswap [1], which has gained a significant following since its launch in 2018. In essence, Uniswap is a liquidity pool. Liquidity providers (LPs) deposit two different tokens in a pool,

and the ratio of these tokens determines the price at which the pool's assets can be traded. Whenever a user wants to trade one token for another, they simply swap tokens with the pool at the current exchange rate. The user pays a fee for the transaction, which is distributed to the LPs as a reward for providing liquidity to the pool. Uniswap is based on an automated market maker (AMM) DEX, where the exchange rate of each trade is determined by predefined algorithms and market liquidity reserves [11]. Uniswap V2 [12, 13], launched in 2020, functions as a constant product market maker (CPMM), where liquidity providers supply liquidity on the entire price range.

2.3 Trading Strategies

Decades of asset trading on traditional exchanges have brought a collection of market manipulation techniques, such as front-running [14], pump and dump schemes [15], and wash trading [16]. These trading strategies used on centralized exchanges bear the potential to be used on their decentralized counterpart. The transparency of blockchain transactions allows for the emergence of a new opportunity for arbitrageurs on decentralized exchanges (DEX) [17]. The emergence of potential trading opportunities is especially evident when dealing with pending transactions on a blockchain network. Once a user initiates a transaction, it is transmitted to network nodes for validation and then propagated to miners. At this point, the transaction is visible to all, yet it remains unconfirmed. This provides an opportunity to deploy trading strategies based on this information, which can lead to advantageous outcomes once the transaction is finally confirmed.

When front-running, a trader takes advantage of the knowledge of a pending transaction to profit from it before the transaction is executed. For example, if a trader knows that a large buy order is going to be executed soon, they may buy the same asset beforehand and then sell it to the original buyer at a higher price [14]. Similar to this strategy, back-running [18] entails taking advantage of the knowledge of a pending transaction to profit from it. For example, if a trader knows that a large transaction will be executed, they might be able to take advantage of price discrepancies arising shortly after the completion of this transaction. Pump-and-dump trading [15] strategies involve artificially inflating the price of an asset and then selling their positions at a profit once the price has risen. Wash trading [16] is a form of market manipulation in which a trader buys and sells the same asset repeatedly in order to create the illusion of increased trading activity. The goal is to attract other traders to the market and drive up the price. Cyclic arbitrage [11, 19] is a trading strategy in which a trader exploits price discrepancies between three different currencies in order to make a profit. Boonpeam et al. [20] analyzed the most profitable trading routes to perform cyclic arbitrage. Based on these trading routes they build a trading bot

that monitors the price discrepancies of a certain token pair within one DEXs as well as between multiple DEXs and exploits them to make a profit.

2.4 Reinforcement Learning

Machine learning can be divided into three learning methods, namely supervised learning, unsupervised learning, and reinforcement learning [21]. Reinforcement learning (RL) involves an agent learning how to make decisions in an environment by interacting with it and receiving feedback in the form of rewards or penalties.

One of the main advantages of RL is its ability to learn optimal behavior in complex environments where it is difficult to define heuristics for decision-making. RL can learn to play games, navigate robots, optimize resource allocation, and even develop new strategies for trading in financial markets [22, 23, 24]. One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation [25]. To obtain high rewards, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing rewards. But to discover such actions, it has to try actions that it has not selected before and therefore oftentimes might encounter negative rewards. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future [26].

Reinforcement learning has become a popular approach in trading research in recent years. Its application has been explored in various branches such as portfolio management, automated trading, and other areas of quantitative finance [27, 28, 29] . For instance, researchers have investigated the use of RL for portfolio management, as demonstrated in the study by Huang et al. (2022) [30] on deep RL for portfolio management. The study proposes a deep RL algorithm to optimize asset allocation, which takes into account the complex interactions between different assets and their dynamic relationships.

The use of reinforcement learning (RL) in automated trading has emerged as a promising approach to developing trading strategies that can effectively respond to dynamic market conditions. One such example is the use of an adaptive Q-learning algorithm for automated trading in equity stock markets, as demonstrated by Chakole et al. [31]. This algorithm is capable of dynamically adjusting to non-stationary market conditions and incorporates risk management in its decision-making process. Furthermore, RL has been employed in other areas of quantitative finance, such as option pricing and risk management [32].

Although research on employing reinforcement learning (RL) specifically for trading within decentralized finance systems remains limited, RL has shown great promise in a wide range of applications, demonstrating its potential as a powerful tool for learning optimal behavior in complex environments. With the continued advancements in machine learning and artificial intelligence, particularly in

the domain of deep reinforcement learning, it is expected that RL will play an increasingly important role in the development of intelligent systems, including potential applications in decentralized finance trading.

Chapter 3

Material and Methods

3.1 Simulation Environment

In order to train a reinforcement agent, it's necessary to create a simulation of a decentralized exchange system, which is based on an abstracted version of a blockchain mechanism. The purpose of this simulation is to include only the necessary elements for specific experiments, allowing for faster simulation and reducing the complexity that could result in errors. This simulation serves as the learning environment for the agent and forms the basis for evaluating its performance. The simulation is programmed using Python and aims to replicate the functioning of real DEXs by simplifying their processes.

The upcoming section will provide an explanation of the various components of the simulation.

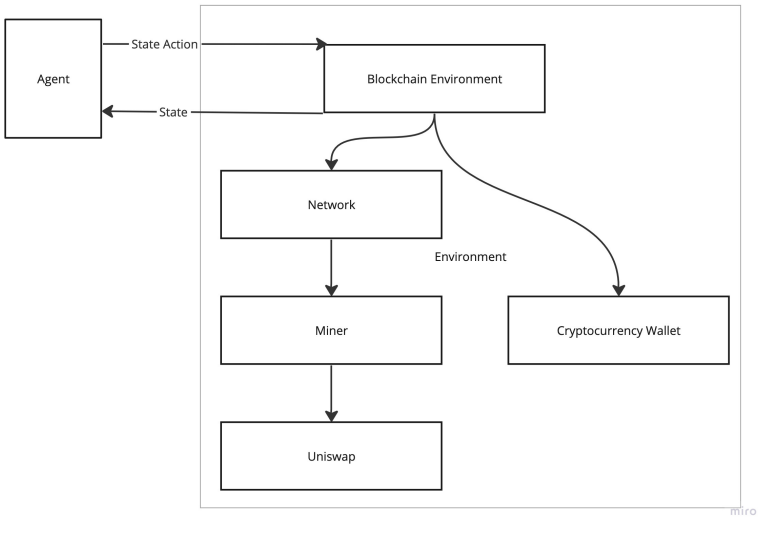


Figure 3.1: Graph of the simulation environment

3.1.1 Blockchain Environment

The blockchain environment constitutes the central module in the simulation, responsible for initializing all critical elements. This environment not only supervises the simulation reset process, which involves reestablishing the reserve values of the liquidity pool and the agent's personal wallet but also controls the simulation's advancement. By processing the agent's state action input, the environment calculates the subsequent simulation state and communicates this to back to the agent. Furthermore, the environment computes the rewards to be allocated to the agent at each simulation iteration. Additionally, it serves as the point of origin for either simulating block data or incorporating real data, both of which comprise transactions from other market participants. The simulation requires several components, including a cryptocurrency wallet, a Uniswap class, a miner class, and a network class.

3.1.2 Cryptocurrency Wallet

In decentralized finance, wallets play a critical role in managing and accessing financial assets. DeFi wallets are non-custodial, meaning that the user maintains full control over their private keys and assets. However, to facilitate the simulation, we will introduce a centralized unit that manages all the wallets. This will be simulated by a cryptocurrency wallet class, which will oversee and execute cryptocurrency transfers throughout the simulation.

3.1.3 Uniswap

The availability of Uniswap v2's smart contract code on the Ethereum blockchain allows us to reconstruct a simulation of the platform. By analyzing the code, we can gain a thorough understanding of how the platform operates and then create a simplified version of it. First, we will define the key parameters of the protocol, such as the liquidity pools, the token pairs, and the fees. Secondly, the Uniswap class must simulate the behavior of the protocol, including the process of adding liquidity, swapping tokens, and calculating fees. The subsequent formulas are derived from the paper [11].

Constant product formula

The constant product formula is a widely used pricing mechanism that determines the exchange rate of two tokens in a liquidity pool . The formula states that the product of the number of units of each token in the pool remains constant. The following equation holds during the transaction:

$$a \times b = (a + \Delta_a \times r1) \times (b - \Delta_b \times r2) \tag{3.1}$$

where a and b represent the number of units of the two tokens in the pool, and Δ_a and Δ_b represent the amount of tokens to be added or removed from the pool. $r1$ and $r2$ are values use to calculate the fees of the transaction.

$r1$ and $r2$ are defined as 0.997 and 1 respectively, indicating that the token added to the pool will experience a 0.3% loss in value and the other token will not experience any change in value. This value is the transaction fee paid to the liquidity providers.

Swapping Coins Swapping means exchanging an amount of cryptocurrency a for the resulting amount of cryptocurrency b or vice-versa. The amount received follows the constant product formula as well as the liquidity fee deduction.

Swap a for b :

$$delta_b = \frac{r1 * r2 * b * delta_a}{a + r1 * delta_a} \tag{3.2}$$

Swap b for a:

$$delta_a = \frac{r1 * r2 * a * delta_b}{b + r1 * delta_b} \tag{3.3}$$

Adding Liquidity Adding liquidity to the DEX means becoming a liquidity provider of the specific liquidity pool by depositing coins from both cryptocurrencies involved. The amount of coins added will be proportional to the reserves in the liquidity pool. The computation follows the formula:

$$delta_l = l * \frac{delta_a}{a} \tag{3.4}$$

$$l = \sqrt{ab} \tag{3.5}$$

When a trade is executed in a liquidity pool, a fee is charged, which is split between the liquidity providers who deposited tokens in the pool. This incentivizes liquidity providers to deposit funds and receive shares of the fees generated by trading activity.

When a liquidity provider is involved in a specific liquidity pool, they are able to withdraw their amount of liquidity tokens from the pool. The percentage of liquidity to be given to the trader is determined by the variable δ_l. To calculate the balance of the two coins received, the following formula can be used:

$$\delta_l = l \cdot \frac{\delta_a}{a} \tag{3.6}$$

Here, l represents the current balance of liquidity tokens in the pool, a represents the current balance of one of the tokens in the pool, and δ_a represents the desired change in the balance of that token. The same formula applies to the second coin in the pool, denoted by b.

3.1.4 Miner

The miner is a crucial component of the Ethereum blockchain. It verifies and adds new transactions to the blockchain, and in return receives rewards in the form of Ether. In the simulation, the pool of miners for the Ethereum blockchain is fused to just a single miner, who handles the transactions of the specific liquidity pool being invested in. This miner is responsible for ordering the execution of the transactions using a greedy sorting algorithm, which assumes that the miner seeks to maximize their profits without regard for the long-term health and stability of the network. While real-world miners may have other ordering strategies, limited resources, and knowledge of the network, this simplifying assumption is still feasible [33].

3.1.5 Network

The network class has a specific role, which is to hold the pending transactions for each step. These transactions are created in the blockchain environment and then added to the network. The miner class is responsible for removing the transactions from the network and mining them into a block.

3.2 Data Collection

To train a model that accurately reflects real-world dynamics, comprehensive real-world data is essential for generating simulations. This requires a large collection of historical data, which can be readily accessed due to the transparency of blockchain technology. The data needed to simulate the interaction with a specific liquidity pool are the following:

- The historical swap, add, remove transactions of the specific liquidity pool
- The historical sync value of the specific liquidity pool (reserves)
- The historical centralized market prices

The decentralized data is already gathered on a database server of ETH and could be queried from there directly. The data encoded in hex values then needed to be split into the relevant parts as well as encoded into the different types of

transactions. We opted to analyze a range of blocks from 6,000,000 to 12,000,000 as this time period corresponds to when Uniswap V2 was the predominant exchange platform. This decision was made because we wanted to exclude Uniswap V3 and focus solely on the use of Uniswap V2. The centralized data which consists of the centralized coin price at specific time steps can be downloaded from Yahoo Finance using the timestamp from the block mining process. The dataset has been divided into two separate files. The first file comprises transaction data, which includes the type of transaction and the corresponding gas price 3.1. The type is encoded directly using the encoding scheme that will be employed in the simulation. The transactions are represented by an array containing various values such as block number, amount, reserve0, and reserve1. The second file consists of balance data, which is represented by an array containing block numbers, truncated timestamps, priceC0, priceC1, reserveC0, and reserveC1 3.2. The centralized market prices of the selected coins were collected using the block mining timestamps.

Transaction Type	**Amount**	**Gas Price**	**Block Nbr**
1 (swap0f1)	16800000000	13000000000	10092378
-1 (swap1f0)	22000000	33000000000	10094587
2 (add)	2661682	19000000000	10096242
-2 (remove)	162488585	13000000000	10098471

Table 3.1: First Data File

Block Nbr	**PriceC0**	**PriceC1**	**ReserveC0**	**ReserveC1**
10092378	1.001	1.001	16801112910	16798896494
10093076	0.995	0.996	16803677185	16796340640
10093368	0.995	0.996	16624266862	16978170165

Table 3.2: Second Data File

The dataset comprises data of different types depending on the category or coin represented. The gas price is always in GWEI, and the amount is expressed in the specific unit of the cryptocurrency, with a value of 10e-6 for USDT and USDC and 10e-18 for ETH. Upon generating the dataset, essential aspects were examined, such as the maximum number of transactions per block, which is vital for defining an accurate observation space. Additionally, the mean number of transactions per block is utilized to determine the query size in the attention network implementation, as referenced in 5.3.2. This value is 18 for the stable coin dataset and 64 for the unstable coin dataset.

3.3 Reinforcement Learning

This section consists of three key components: architecture, algorithm, and reward. The architecture refers to the overall framework of the system that will perform the backpropagation. The algorithm section outlines the specific learning methods and techniques used to optimize the system. Finally, the reward section defines the objective of the system, as well as the specific rewards or penalties that will be used to guide the learning process.

3.3.1 Architecture

The basic architecture of a reinforcement learning model consists of three main components: the environment, the agent, and the reward structure. The environment is an external system that the agent interacts with and receives feedback from. The agent is the learning system that takes actions based on the current state of the environment and the feedback received from the environment in the form of a reward signal. The reward signal is a value that the agent receives from the environment after each action, which indicates the quality of the action in terms of how well it aligns with the agent's objective. The agent's goal is to learn a policy, which maps states to actions while maximizing the cumulative reward signal over time. For our specific use case, we will use an Actor-Critic architecture consisting of two components for modeling the agent. The functioning of these two components will be explained in more detail in the next section.

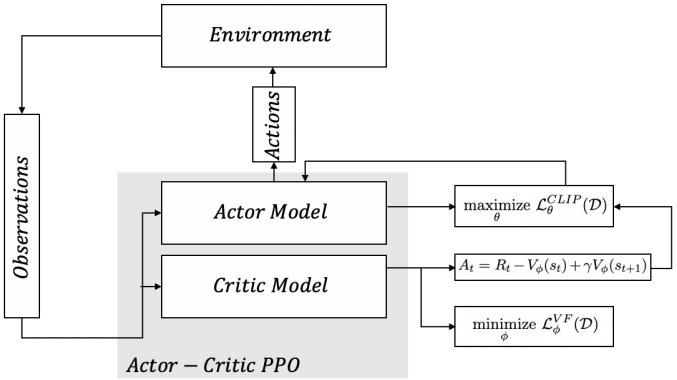


Figure 3.2: Actor-Critic Architecture PPO

3.3.2 Algorithm

A popular RL algorithm is Proximal Policy Optimization (PPO) which was published by OpenAI in 2017 [34]. PPO is a policy gradient algorithm that is designed to strike a balance between sample efficiency and ease of implementation. PPO has been shown to outperform several other state-of-the-art RL algorithms on a wide range of benchmark tasks. It is based on the actor-critic architecture, where the actor learns to map states to actions and the critic learns a value function to predict the expected reward [35]. PPO aims to simplify the approach of Trust Region Policy Optimization (TRPO) [36] while achieving similar performance. Compared to other policy gradient algorithms, TRPO shows favorable data efficiency. However, solving higher-order optimization problems is computationally expensive. To tackle this issue, PPO introduced a clipped surrogate objective only requiring first-order optimization and thus reaching higher sample efficiency.

Actor Optimization:

$$\underset{\theta}{\text{maximize}}\ \mathcal{L}_{\theta}^{CLIP}(\mathcal{D}) \tag{3.7}$$

where:

θ is the parameter of the policy $\mathcal{D}$ is the set of transitions (s_t, a_t, r_t, s_{t+1}) collected during training $\mathcal{L}_{\theta}^{CLIP}$ is the clipped surrogate objective function for the policy, defined as:

$$\mathcal{L}^{CLIP}\theta(\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum(s_t, a_t, r_t, s_{t+1}) \tag{3.8}$$

$$\in \mathcal{D}\min\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}A_t, g(\epsilon, A_t)\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\Big) \tag{3.9}$$

where:

$\pi_{\theta_{\text{old}}}$ is the policy used to generate the data during the previous iteration $g(\epsilon, A_t)$ is a function that clips the advantage estimate A_t to be within the range $[1-\epsilon, 1+\epsilon]$. ϵ is a hyperparameter that controls the extent of the clipping.

The actor optimization aims to maximize the clipped surrogate objective function $\mathcal{L}_{\theta}^{CLIP}$, which encourages the policy to increase the probability of actions that lead to higher advantage estimates A_t, while limiting the size of policy updates using the clipping function $g(\epsilon, A_t)$.

Critic Optimization:

$$\underset{\phi}{\text{minimize}}\ \mathcal{L}_{\phi}^{VF}(\mathcal{D}) \tag{3.10}$$

where:

ϕ is the parameter of the value function $\mathcal{D}$ is the set of transitions (s_t, a_t, r_t, s_{t+1}) collected during training $\mathcal{L}_{\phi}^{VF}$ is the mean squared error loss for the value function, defined as:

$$\mathcal{L}^{VF}\phi(\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum (s_t, a_t, r_t, s_{t+1}) \in \mathcal{D}\Big(V_\phi(s_t) - \hat{V}_t\Big)^2 \tag{3.11}$$

where:

$V_\phi(s_t)$ is the estimated value of state s_t. $\hat{V}_t$ is a target value for the state-value function, typically set to the discounted sum of future rewards.

3.3.3 Reward

In this system, the reward for the agent is calculated based on the change in its wealth resulting from taking an action. The cryptocurrency wallet and liquidity tokens held by the agent are evaluated before and after the action, and the difference between their values is computed. To standardize the comparison, all assets are converted into USD using the historical centralized market price. This change in wealth is then used to determine the agent's reward.

$$\text{reward} = (p_{c0_{cex}} \cdot \Delta(w_{c0} + lp_{c0})) + (p_{c1_{cex}} \cdot \Delta(w_{c1} + lp_{c1})), \tag{3.12}$$

where $\Delta(x) = x_{\text{new}} - x_{\text{old}}$ is the change in value of a variable x, and w_{c0}, lp_{c0}, w_{c1}, lp_{c1} are the wallet and liquidity pool balances of tokens c_0 and c_1.

3.4 Performance analysis

The main metric for performance analysis is the summed numerical reward received by the agent. This value tells us how much profit the agent generated over the testing time span. The performance of our agent needs to be compared to specific baselines to assess its relative performance. The computation of the baselines will be explained in the section 5.4.3.

3.5 Risk assessment

The assessment of risk for the assets held by the agent, both in the liquidity pool and in its possession, is essential. A significant risk associated with being a liquidity provider is the potential for imbalances in the liquidity pool, where one asset has considerably more liquidity than the other. Such imbalances can lead to slippage between the two assets, possibly affecting the agent's profitability, as

it might receive less value than initially invested. This risk is commonly known as impermanent loss for liquidity providers [37].

Another risk that the agent encounters concerns the balance of its portfolio after a specific period in the simulation. If the agent begins with equal amounts of both coins, any shifts in the coin ratios observed after the testing period can be seen as a loss in diversification. This loss in diversification is known to be riskier, as it increases the market risk associated with the particular coin that is receiving a higher allocation in the portfolio [38]. It is worth noting that the risk assessment carried out in this thesis has limitations. Additional risks, such as those arising from trading bots, miner extraction, or operational risk, are not covered in this thesis.

Chapter 4

Experiments and Results Simulation Environment

The first set of experiments was conducted on a down-scaled version of the simulation which entailed artificially generated block data. This first range of experiments is done in order to test the functionalities of the reinforcement learning algorithm before using real data. The simplification consists of :

- Trader modeling
- Centralized price modeling
- Liquidity pool reserve modeling

4.1 Trader modelling

In the context of trader modeling, a key assumption is made: traders do not engage in actions or respond to the agent's actions. This simplification is adopted to minimize complexity, as training active traders would extend beyond the scope of this thesis. The transaction data model is grounded in real-world data. Transaction type distribution is ascertained through weighted sampling 4.1, while the transaction amount is uniformly sampled from the observed ranges in actual data 4.2 .

Actions	SWAP0F1	SWAP1F0	ADD	REMOVE
Probabilty	0.4	0.4	0.1	0.1

Table 4.1: Sampling probability of actions.

Actions	amount	gas price
Range	0.1 - 100000 USD	0.1- 280 USDC

Table 4.2: Ranges for the transaction amount and gas price

4.2 Price modeling

The centralized price is derived from a modeling method incorporating historical data, using Geometrical Brownian motion [39]. Wiener process [40], also called Brownian motion, is a stochastic process that models the random fluctuations in the price of an asset over time. It assumes that the asset's price changes are independent and identically distributed, with a normal distribution and a mean of zero. Geometric Brownian motion extends the Wiener process by adding a drift component, representing the asset's expected growth rate. The drift component is multiplied by the current price of the asset, resulting in a process that is proportional to the asset price. The formula for Geometric Brownian Motion is given by:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \tag{4.1}$$
$$W_t \sim \mathcal{N}(0, t)$$

where S_t represents the stock price at time t, μ is the drift parameter, σ is the volatility parameter, W_t is a Wiener process, and dt represents a small change in time.

4.3 Liquidity pool modeling

The liquidity pool is modeled by employing the inversely proportional ratio of the simulated centralized market price (refer to Section 4.2) and a quantity that resembles the one associated with the selected Uniswap pair (in this case, using a value on the order of 10^8).

$$\frac{reserve_{dex_a}}{reserve_{dex_b}} = \frac{price_{cex_b}}{price_{cex_a}} \tag{4.2}$$

4.4 Observation Space

The observation space, also known as the state, represents the current network status provided to the agent at each step. It contains the necessary information for the agent to determine its next action based on its policy. This information is essential for the learning process and typically includes:

- **LP Reserve Information**: This refers to the liquidity pool (LP) reserves, which the agent must consider when calculating the correct pricing for transactions.
- **Centralized Market Prices**: These are the current market prices of cryptocurrencies outside of the decentralized exchange. This information is crucial for the agent to evaluate the liquidity pool's pricing and determine whether a given trade is profitable.
- **Pending Transactions Information**: This refers to all the transactions that are currently in the queue waiting to be processed. The agent needs this information to determine the optimal timing for placing a trade.

The full observation, therefore, contains the following information :

$$o = (p_{c0}^{cen}, p_{c1}^{cen}, p_{c0}^{dec}, p_{c1}^{dec}, t_{1,type}, t_{1,gas}, t_{2,type}, t_{2,gas}, \ldots, t_{n,type}, t_{n,gas})$$

where p_{c0}^{cen} and p_{c1}^{cen} represent the centralized prices of assets $c0$ and $c1$, respectively, and p_{c0}^{dec} and p_{c1}^{dec} represent the decentralized prices of assets $c0$ and $c1$, respectively. The remaining variables $t_{i,type}$ and $t_{i,gas}$ represent the type and gas price of each pending transaction i, with n representing the total number of pending transactions.

By incorporating all of this information into the observation space, the agent is equipped to make informed decisions about when and how to execute trades on the decentralized exchange.

4.5 Action space

The actions involved in a transaction consist of the type of transaction and the positioning of the transaction relative to the pending transactions. The transaction type can be easily represented using a categorical value, while the positioning is based on the gas price attached to the transaction, which is a continuous value. The graphic below illustrates the positioning possibilities at each step.

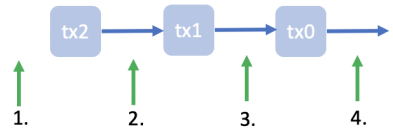


Figure 4.1: Visualization of position encoding

In Reinforcement Learning (RL) algorithms, the action space can be either discrete or continuous, depending on the nature of the task. However, since our

model requires both continuous and discrete components, a specific encoding is necessary to enable the use of a fully discrete or a fully continuous model. As a result, we conducted two sets of experiments, one using discretized actions and the other using only continuous actions to encode the components. The first approach involved discretizing the continuous gas price variable into a set of distinct values which mirror the positioning. This resulted in a fully discrete action space that could be easily used with standard RL algorithms. The second approach involved using a continuous gas price variable and encoding the type of the transaction into a continuous variable resulting in a fully continuous action space.

4.6 Discrete Model

For our first experiment, we employed a discrete action space to encode information. Specially, we used a multidiscrete action space where the components are represented as a tuple $a = (a_1, a_2)$ with a_1 and a_2 taking on discrete values. Specifically, a_1 encoded five different transaction types, while a_2 encoded the agent's transaction positions relative to other transactions which for n transaction translates to $n + 1$ possible positions. As a simplification, we will work with a constant number of transactions and set $n = 4$ for all experiments in the simulation environment experiments.

$$a = (a_1, a_2) \quad \text{where} \quad a_1 \in 0, 1, 2, 3, 4, \quad a_2 \in 0, 1, 2, ...n$$

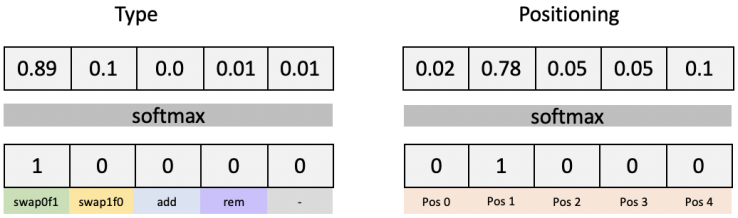


Figure 4.2: Action space mapping of the discrete model.

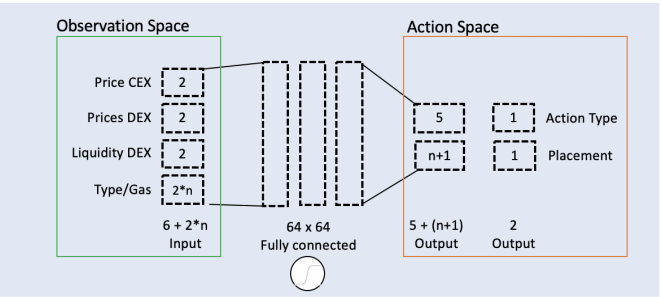
The encoding of the type variable is optimal since it represents a categorical value, while the encoding of the position is not ideal, particularly for a larger number of pending transactions. The positioning requires a fixed number of transactions and expands with the transaction count, making this model impractical for variable transaction numbers and increasing transaction volumes.

4.6.1 Reward shaping

The initial reward, which reflects the post-action wealth difference of the agents, is shaped using the arctan function with chosen parameters. One reason to use the arctan function for reward shaping is that it can provide a smooth, continuous reward signal that can be easily scaled and shifted to reinforce a certain behavior. Another reason to use the arctan function for reward shaping is that it has a natural saturation point which helps to stabilize the reward back-propagation to the agent. This saturation can also increase exploration of the agent as the reward differences will be flattened out. A disadvantage is that the distinction between receiving good rewards versus excellent rewards becomes diminished.

4.6.2 Network Architecture

The network architectures used for the discrete model experiments are small-scale neural networks. The actor-network and the critic-network utilize a `[64, 64]` fully connected layer architecture with a *tanh* activation function.



Specifications	
Algorithm	PPO
Architecture	[64, 64] fully con.
Activation function	tanh
Obs. Space	$6 + 2 * n$
Action Space	$5 + (n + 1)$
Reward	$atan(diff_{wallet})$ * 2/pi

Figure 4.3: Network architecture visualization discrete model

4.6.3 Hyperparameter Choice

The hyperparameter tuning was conducted using the rlzoo implementation from stable-baselines-3 [41]. The final hyperparameters can be found in this table A.1. After figuring out the most rewarding parameter some minor manual changes were conducted concerning the entropy coefficient. These minor changes will be explained in the learning pipeline below 4.6.4.

4.6.4 Learning Pipeline

To prevent the agent from getting stuck in one particular behavior (non-optimal), it is necessary to train the reinforcement model in several stages. The learning pipeline comprises four stages. The first stage utilizes a masking mechanism that

restricts the agent from using the withhold action, which promotes more thorough exploration. During the initial training phase, the agent may encounter strong negative rewards that hinder optimal learning. Therefore, the agent tends to converge to the action where it does not receive negative rewards. The second stage involves an increase in entropy and reintroducing the withhold action. In the third stage, the objective is to eliminate any remaining negative reward behavior. To achieve this, the penalty for negative rewards is upscaled by a factor of 10. The final stage involves selecting the model state that exhibits the desired behavior. This is accomplished by evaluating each saved model state on 20 random data batches.

4.6.5 Results

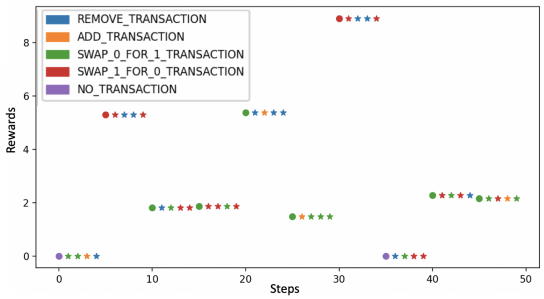
Using the learning pipeline above multiple experiments were conducted. Initially, the simulation was initiated with a liquidity pool balance that matched the CEX prices, which was then modified by the behavior of the traders and agents. In this setting, the agent's behavior shows a strong ability to learn about reserve imbalances, but no ability to learn about pending transactions 4.4a . This behavior is motivated by the fact that the traders create strong imbalances due to their randomness which then allows the agent to focus on exploiting them easily. To counteract this unwanted behavior a second experiment was conducted which consisted in resetting the balances after each step. Therefore the agent is compelled to focus exclusively on the pending transaction to produce favorable rewards. Although the agent's actions type aligns with types of pending transactions, the positioning is still sub-optimal 4.4b . The third experiment overcame this limitation by encoding traders' transactions with position indexes based on their gas prices in descending order. This allowed the agent to understand the optimal trading behavior and streamline the learning process for optimal positioning 4.4c . In Figure 4.4, we compare the results of three different experiments.

4.7 Continuous model

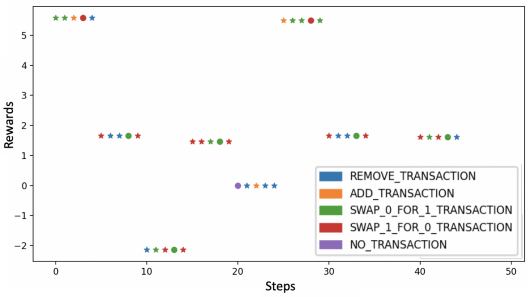
Conversely, we have developed a model that employs a continuous action space. To simplify the process, the number of transactions is held constant at 4. These action values are depicted as continuous values, which can be formulated using the following equations:

$$a = (a_1, a_2) \quad \text{where} \quad a_1 \in [-1, 1], \quad a_2 \in [-1, 1]$$

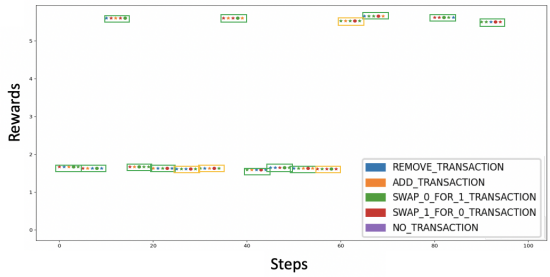
In this case, a_1 denotes the action type, which is modeled using three categories: a range for *swap0f1*, a range for *swap1f0*, and a range for *withhold*. The range $[-1, 1]$ is employed to represent these categories, where the range $[-0.1, 0.1]$ signifies



(a) Experiment 1: no placement learned



(b) Experiment 2: random placement learned



(c) Final Experiment: semi-optimal positioning learned (green=optimal placement, yellow=sub-optimal placement)

Figure 4.4: Comparison of experiment results. The reward analysis is visualized using a plot that combines several types of information at each step: the reward value of the agent, the transaction type of the agent, and the transactions of other parties and their types. To display all this information in a clear way, the following visual encoding is used: the x-axis represents the step number, the y-axis represents the reward value, and colors are used to differentiate the transaction types (swap0f1, swap1f0, add, remove, withhold), and the shape of the scattered points is used to indicate whether the transaction came from the agent or the traders (Dot = Agent, Star = Trader).

withholding, $(0.1, 1]$ corresponds to *swap0f1*, and $[-1, -0.1)$ represents *swap1f0*. The second component a_2 encodes the action positioning using a direct prediction of the gas price. Again, the range $[-1, 1]$ is used to represent this component, where -1 represents a low gas price and 1 represents a high gas price. To map the first continuous value to 3 discrete actions, we can use the following function:

$$f(a_1) = \begin{cases} s1f0, & \text{if } a_1 \in [-1, -0.1) \\ withhold, & \text{if } a_1 \in [-0.1, 0.1] \\ s0f1, & \text{if } a_1 \in (0.1, 1] \end{cases}$$

For the second continuous variable, we can interpolate it to a value between 0 and 10 using the following formula:

$$gas_{price} = \frac{a_2 + 1}{2} \times 10$$

This maps the range of [-1, 1] to [0, 10].

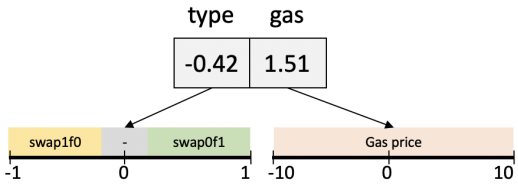


Figure 4.5: Visualization of the action space mapping continuous model

The encoding is optimal for the positioning as it is able to directly output the gas price whereas it becomes ambiguous for the type choice. The encoding of the types using a spectrum that associates specific ranges to actions might encumber the optimization process.

4.7.1 Reward shaping

To shape the rewards, an *arctanh* function was applied to the wealth difference in cases where negative rewards were required, while a simple identity mapping was used for positive rewards. The rationale behind this approach was to enable the agent to distinguish between moderate and good actions. Therefore, it was crucial that positive rewards did not saturate to allow the agent to perceive the differences in quality between different positive outcomes.

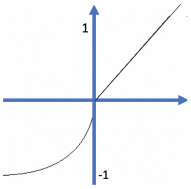
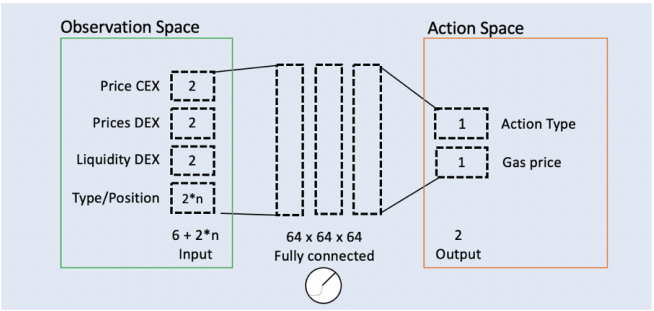


Figure 4.6: Reward shaping continuous model

4.7.2 Network architecture

The architecture of the continuous network is similar to that of the discrete one, with the only difference being an additional layer of depth in the neural network.



Specifications	
Algorithm	PPO
Architecture	[64, 64, 64] fully con.
Activation function	tanh
Obs. Space	$6 + 2 * n$
Action Space	2
Reward	– Negative values : $atan(diff_{wallet})$ * 2/pi – Positive values : $diff_{wallet}$

Figure 4.7: Network architecture visualization of the continuous model

4.7.3 Hyperparameter Choice

Due to the continuous model stagnating at a moderate performance level, a more comprehensive analysis of feasible parameter/hyperparameter is required. In the process of tuning hyperparameters, we employed a combination of grid search and insights from relevant papers [42, 43]. An aspect left unexplored in the discrete model experiment is space normalization. Space normalization is a vital aspect of reinforcement learning [44], as it ensures that observations, actions, and rewards are scaled within a consistent range of -1 to 1. This facilitates the learning process of the agent by allowing the neural network to better capture relevant features and gradients, enabling the agent to learn a good policy. Normalization also aids in generalizing the learned policy to new tasks or environments, as the agent can adapt more effectively without requiring a complete retraining. The final hyperparameters can be found in this table A.1.

Additionally the gSDE (generalized State-Dependent Exploration) is a powerful tool that can enhance stability and performance during reinforcement learning [45].

$$a_t = \pi_\theta(s_t) + \sigma(s_t) \odot \epsilon_t \tag{4.3}$$

Here, the different variables represent:

- a_t: The action taken by the agent at time t.
- $\pi_\theta(s_t)$: The deterministic policy function, which maps the current state s_t to an action, parameterized by θ.
- $\sigma(s_t)$: The state-dependent exploration noise function, which maps the current state s_t to a noise scaling factor.
- $\odot$: The element-wise (Hadamard) product.
- ϵ_t: A noise vector sampled from a zero-mean Gaussian distribution at time t.

In this formula, the deterministic policy function π_θ is used to determine the action to take in a given state, while the state-dependent exploration noise function σ adds randomness to encourage exploration. The element-wise product of $\sigma(s_t)$ and ϵ_t allows the agent to explore more effectively by adapting the noise scale based on the current state. By adding noise to the gradient update step, gSDE can regularize the updates, prevent overfitting, and encourage exploration of the action space.

Furthermore, increasing the batch size can lead to better results [46], as it enables the agent to learn from a larger and more diverse set of experiences. Larger batch sizes can also provide more stable updates to the neural network parameters, preventing overfitting to small batches of data.

4.7.4 Learning Pipeline

The learning pipeline used for the continuous model is the same as for the discrete model 4.6.4.

4.7.5 Results

The continuous model was subjected to a series of comprehensive hyperparameter tuning experiments to evaluate its performance. The final hyperparameters can be viewed in this table A.1. As a result of these experiments, it was observed

that the model exhibited an trading accuracy of around 90 percent. This level of performance was deemed sufficient and paved the way for the model to be upscaled to train on real historical data in a simulation environment. This would allow the model to be trained and evaluated using actual market data, which is critical for establishing its real-world applicability and effectiveness.

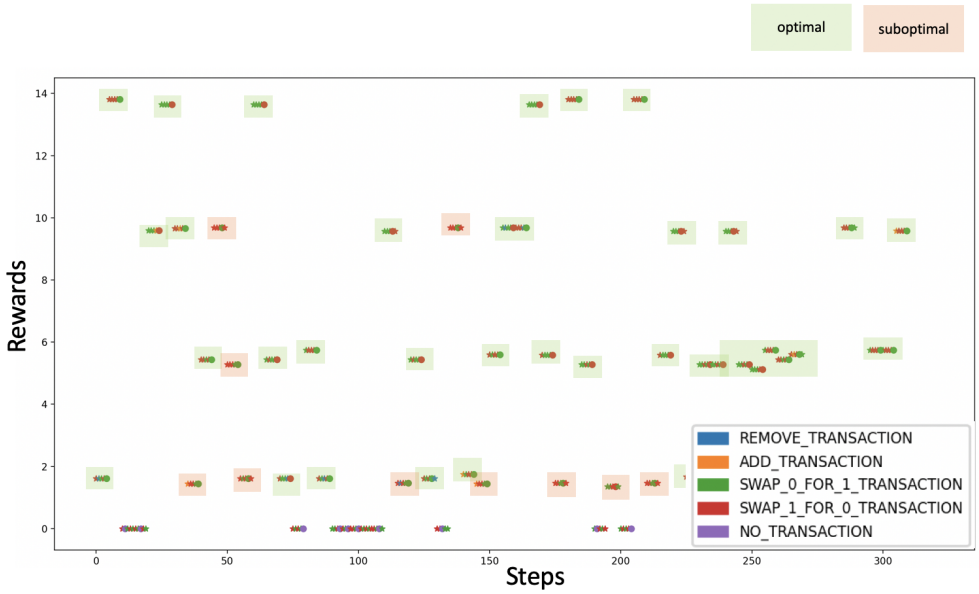


Figure 4.8: Visualisation of the evaluation output of the continuous model

Chapter 5

Experiments and Results Historical Data

Although semi-randomized data is useful for experimenting with the simulation's structure and developing a training pipeline, it is common practice to use historical data to prepare the agent for real-life trading situations [47]. This procedure is called back-testing [34] and is frequently used to evaluate investment and trading strategies. While the motivation behind this method is to apply a trading behavior that was successful in the past to future data, it is important to be cautious about assuming that strategies that worked on past data will perform well on future data. One should also be mindful of the survivorship bias that can arise when using historical data. Survivorship bias occurs when the focus is placed on the surviving entities in the historical data while disregarding those that have failed [48]. In our case, this can manifest as only considering traders' transactions that were successfully mined into the blockchain, while ignoring those that were not mined due to factors such as network congestion, insufficient gas price or gas limit, sender-initiated cancellations, or other issues [49, 50]. This bias can lead to an inaccurate understanding of the market dynamics and an overestimation of the potential for success of the RL agent.

To integrate historical data with reinforcement learning, it is essential to assume that the actions of our agent will not influence the actions of other traders. We will call this the zero impact assumption. To uphold this assumption, it is imperative to restrict the amount of money traded by the agent to prevent any substantial impact on the liquidity pool balances. If the agent's trades alter the balances significantly, other traders may exploit any arbitrage opportunities that arise as a result. Therefore, keeping the agent's trades low is essential to minimize the chances of such potential exploitation. The assumption of zero impact has implications for the gas price set in the simulation. It is not feasible to assume zero impact and use actual gas prices at the same time since high gas fees [51] for a small amount traded would render any margin for profit non-existent. To overcome this issue, the gas fees for all liquidity pool transactions are divided by a factor of 100, which enables the agent to receive a positive reward and facilitates

learning.

5.1 Spaces

5.1.1 Observation Space

The observation space is constructed using the pending transaction (type, amount, gas price) and the ratio of the DEX (decentralized exchange) reserves, as well as the inverse ratio of the CEX (centralized exchange) prices. This can be formulated in the following equation:

$$o = (p_{c0}^{cen}/p_{c1}^{cen}, res_{c1}^{dec}/res_{c0}^{dec}, t_{1,type}, t_{1,amount}, t_{1,gas}, \ldots, t_{n,type}, t_{n,amount}, t_{n,gas})$$

Due to the variability of transactions per block, the observation space needs to be flexible enough to handle variable shapes. To address this issue, two approaches were experimented with: data padding 5.3.1 and using a network with an attention mechanism 5.3.2, [52]. In addition to handling variable shapes, it is important to normalize the observation space to reduce the variability in transaction amount and allow for optimal performance of the agent. Moving average normalization is a common technique used in reinforcement learning for this purpose [53, 54]. Moving average normalization involves calculating a rolling average of the input data over a fixed window of time. The size of the window is set to the common choice of 100 time steps. The rolling average is then subtracted from each data point, and the result is divided by the rolling standard deviation. Compared to traditional normalization techniques that scale input data to have a mean of zero and a standard deviation of one, moving average normalization is more effective when input data is highly skewed or contains outliers. This can be encountered in the dataset as the amount involved in the transaction can vary significantly. Furthermore, the rolling average and standard deviation are calculated only over the training data and are applied to the validation and test sets using the same values to ensure that the normalization process does not introduce any bias into the evaluation of the model.

5.1.2 Action Space

The action space is continuous, as in the continuous model of the simulated data environment. Nevertheless, the model must be scaled up to accommodate not only swap and withhold actions but also add and remove actions. Additionally we want the agent not just to be able to predict a value for the gas price but also to set the amount it wished to proceed in the transaction. We opted for an approach, which maps the action types onto two separate ranges within $[-1, 1]$; one for the swaps and another for the add-remove actions which we will call the liquidity provision (lp) range. This doubling of action space variable needs

to be processed for both the transaction gas price and the transaction amount, resulting in an action space predicting 6 values. Consequently, our agent is able to place two transactions per block, allowing for all the actions to be chosen. Withholding is achieved by selecting an amount of zero and, therefore, does not need to be encoded within the type ranges.

Therefore at each step the action choice of the agent will contain:

$$\mathbf{a} = (type_{swap}, amount_{swap}, gasPrice_{swap}, type_{lp}, amount_{lp}, gasPrice_{lp})$$

The encoded values for the amount and the gas price need to be interpolated to the real data ranges. The gas price is mapped from the interval $[-1, 1]$ to $[0, \max(\text{gas price in dataset})]$, where max(gas price in dataset) represents the highest gas price found in the dataset. This decision is made to enable the agent to potentially front-run any transaction. In a similar manner, the transaction amount is transformed from the range $[-1, 1]$ to $[0, 100 \text{ USD}]$ (zero impact assumption 5).

5.2 Reward shaping

5.2.1 Short Time Rewards

The short-time reward is the reward collected at each step. It is based on the wealth difference of the agent before and after the action as in 4.7.1.

5.2.2 Long Time Rewards

In addition to the regular rewards, the historical data implementation can include long-time rewards that are given to the agent after a certain number of steps. The first such reward was given to encourage more liquidity provision when the model was focused solely on swap strategies. This reward was given after n steps and reflected the increase in wealth due to fees earned over that time span.

A second long-time reward is used to incorporate risk considerations into the training process. An imbalance in the agent's portfolio can decrease diversity and increase market risk. Therefore, after 16 steps, an additional negative reward is computed to penalize the agent if the balance of its wallet is imbalanced. This reward is calculated by taking the negative value of the absolute difference between the ratio of the wallet coins and the inverse ratio of the centralized prices of the coins.

$$\text{Risk Penalization} = -\left|\frac{\text{AgentC1Wealth}}{\text{AgentC0Wealth}} - \frac{\text{CurrentPriceC0}}{\text{CurrentPriceC1}}\right|$$

5.3 Network Architecture

The network structure can be implemented using two approaches to accommodate dynamically changing input sizes. Both networks utilized in this study are based on a neural network with depths of [128, 128, 128] and a tanh activation function. However, they differ in the implementation of their first layers.

5.3.1 Data Padding Architecture

One straightforward approach is to employ a large input that can accommodate the highest count of transactions observed in the training/test data. Specifically, in the case of the USDT-USDC cryptocurrency pair, this count is limited to 18, whereas for the ETH-USDC pair, the count can reach up to 64. The value holders which do not contain a pending transaction are set to 0. The expectation is that the neural network is able to discern the padded data from the relevant data.

5.3.2 Attention architecture

To better handle fluctuations in the length of input data, a more sophisticated approach is to use an attention mechanism that is invariant to input length. This approach involves training a query mechanism in parallel, which selects a fixed number of transactions at each step that is most relevant for the agent to make decisions based upon [52]. The concept behind selecting relevant data using the multihead attention layer is to identify the transactions that have the greatest impact on the agent's potential profit. This means that the network would, for instance, prioritize selecting transactions with a higher amount over those with a smaller amount. By employing multiple heads, the model can learn various connections between the input components, generating diverse attention scores that are subsequently merged. This approach assures a more comprehensive and varied depiction of the input, enhancing the model's ability to capture essential features.

Typically, the initial step in utilizing attention networks involves generating an embedding. However, in cases where the input is already a numerical encoding, this step can be skipped. After this, we need to define the format of the embedding. This was chosen to be of the form `[BATCH_SIZE, NBR_HEADS, 3]`. The last dimension (`3`) encapsulates the number of elements needed for each pending transaction, the `BATCH_SIZE` is the same as set in the network training hyperparameters and the number of heads (`NBR_HEADS`) will be chosen to be a minimum of 4 and then upscaled in further experiments for more complex information grasping. The rationale for selecting 4 as the number of heads is to ensure that the network receives information from at least 4 transactions.

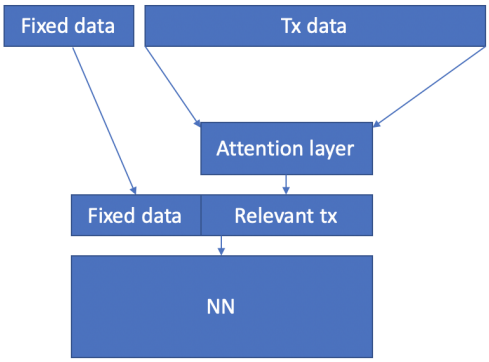


Figure 5.1: Attention Network

This is feasible as most historical data consist of 2-3 transaction per block with only a small fraction containing more than 4 transactions per block. This length will be the dimension of the query vector which will be used to filter out the most relevant data for decision making. Only the transaction data, which is dynamically changing in number, will undergo the query mechanism. On the other hand, the fixed parameters containing the ratios of centralized prices and decentralized prices will be fed directly to the neural network without any modifications.

5.4 Stable Coin Experiment

In the initial set of experiments, a trading pair was used which comprised two stablecoins - USDC and USDT. Stablecoins are digital currencies that are designed to maintain a stable value by being linked to a fiat currency. In this case, both USDC and USDT are pegged to the US dollar, ensuring that their value remains stable relative to the dollar. The decision to conduct experiments on stablecoins was driven by the fact that they are less susceptible to fluctuations caused by centralized market movements. This makes them an ideal candidate for in-depth analysis of trading strategies that involve block-specific arbitrage.

5.4.1 Hyperparameter Choice

Hyperparameter tuning involved using grid search to find the optimal values within predefined ranges. In addition, the entropy coefficient was manually adjusted based on the learning behavior of the agent. The remaining hyperparameters were retained as in the continuous model of the simulated data section 4.7.3.

5.4.2 Data Pipeline

To achieve good generalization, the dataset is divided into training and testing data using an 80/20 split. The training data must be both sequential and diversified at the starting index. This is accomplished by randomly selecting the starting index of each block from a uniform distribution after each reset. Following blocks are executed during the training process until a new starting index

is chosen. This method allows the agent to learn from interconnected data and prevents overfitting to specific imbalances within a particular time span. It is crucial to note that the start-index frequency results in reshuffling the dataset's start-index and resetting all balances, including the liquidity token balance and the agent's wallet balance. This factor is important to consider, as it influences the rewards generated by liquidity pool tokens, with shorter reset frequencies permitting less liquidity provision per reset.

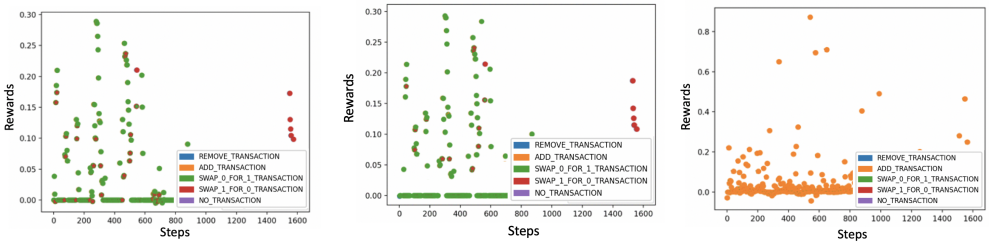
5.4.3 Baselines

To effectively evaluate the agent's performance, we need to compare its results against baseline metrics. These baselines are computed one our simulation and are based on well-known trading strategies 2.3. To ensure the comparability of their performance with that of the agent, these baselines will be given the same amount of currency to invest at each step.

A strategy commonly used on Uniswap is to provide liquidity, as depicted in Figure 5.4c. To accurately compare the rewards for liquidity provision, a baseline was established where the maximum permitted amount of 100 USD was added to the liquidity pool at each step. The resulting reward structure will be employed to evaluate the agent's behavior especially its liquidity provision behaviour.

Building on prior research, an additional trading strategy involves front-running transactions when the balance is favorable for swaps in either direction, as illustrated in Figure 5.4a. This strategy optimizes on the CEX/DEX values to determine the most profitable type of swap and the corresponding amount.

The optimized swapping baseline 5.4b is a swap-based baseline that has been optimized to incorporate the best possible positioning relative to pending transactions. It not only optimizes on the CEX/DEX values but also computes the potential rewards by artificially computing the intermediary states occurring after pending transactions get mined.



(a) Front-running (b) Optimized (c) Liquidity provision

Figure 5.2: Computed Baselines: (a) Result of front-running strategy, (b) Result of optimized swapping strategy, (c) Result of liquidity provider strategy (on chosen test batch)

5.4.4 Experiments and Results

During our research, we conducted several experiments to gain a better understanding of our model's behavior. Out of all the experiments conducted, we identified four primary experiments that had the most significant impact 5.1. All the additional hyperparameter not mentioned in the table or the text can be viewed in this table A.2. They result from the hyperparameter tuning and insights from the previous sections.

Model	Reward Elements	Normalization	Network Architecture	Training steps	Start-index Frequency
trained model 1	short	no	padding	3'000'000	256
trained model 2	short	no	attention	5'000'000	256
trained model 3	short + long (LP)	yes	padding	3'000'000	512
trained model 4	short + Long (LP + Risk)	yes	padding	3'000'000	512

Table 5.1: Overview stable coin experiments

In the initial trial, we utilized hyperparameters that had been fine-tuned, employed short-term rewards without reward normalization, and employed a padding network architecture **(Trained Model 1)**. The training was carried out in two stages: initially, the network was trained for 3,000,000 steps, resulting in nearly optimal outcomes but with a few instances of small negative rewards. Consequently, the network was retrained, with the negative reward being penalized by a factor of ten, which eliminated the negative rewards entirely from our model. Despite the model demonstrating outstanding performance in swaps, being able to learn the optimal positioning we noted that there was a limited level of liquidity provision.

Simultaneously, we conducted a parallel experiment using the attention network **(Trained Model 2)**. The model training was exceptionally steady and generated desirable results, even with a moderate number of steps. However, upon closer inspection, we noticed that the model did not experience negative rewards but failed to capitalize on positioning opportunities. Despite extending the training time, the positioning did not achieve the expected level of optimization. As a result, we pursued experimentation with the padding network in the stablecoin context and halted further exploration of the attention network.

In order to overcome the insufficient liquidity provision, we extended the reset frequency 5.4.2 and implemented a long-term reward system to capture the prolonged advantages of such a provision. As a consequence, we introduced a long-term reward component in our third experiment **(Trained Model 3)**that accounted for the fees earned from liquidity provision over a specified time frame. This supplementary reward (long time reward) 5.2.2 was added to the step-wise reward structure after 16 training steps, resulting in a more diversified transaction choice and a significant improvement in the overall reward generated by the model. Again, to achieve a state of no negative rewards during training and testing, it was necessary to implement a two-stage process, with the second stage

involving a 10-fold increase in the penalization of negative rewards.

As a final experiment, we wanted to explore risk incorporation into the agent's behavior**(Trained Model 4)**. The objective is to explore the agent's capability to incorporate risk versus profit assessment into its strategy through learning. To achieve this, we included an extra long-term reward 5.2.2 that penalizes the overall reward if the agent's personal wallet becomes imbalanced due to previous actions. Similar to the long-term liquidity provision reward, this penalty is added to the step reward after 16 steps. It is subtracted from the reward to penalize instances where the coin ratio deviates from 1, resulting in an imbalanced wallet.

The average metrics for all testing sets are displayed in the table below, as referenced by 5.2. Additionally, a visualization illustrating the resulting agent behavior across different models is available at A.1. This visualization highlights the overall reward distribution and offers a more detailed view of the agent's decisions with respect to pending transactions.

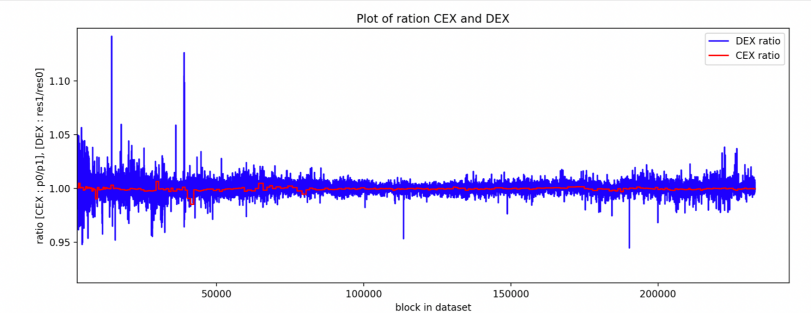
Model	Profit	Amount uniswap c0	Amount uniswap c1	Ratio c0/c1	Reward from LP [usd]	Reward from swap [usd]
lower bound random	-76.31708	1052.60	1054.49	1.01	0.29	-76.61
baseline liquidity provider	17.46	51231.01	51141.51	1.00	18.85	0.0
baseline front-running	20.47	0.0	0.0	0.85	0.0	20.47
baseline optimized swapping	22.92	0.0	0.0	0.82	0.0	22.92
trained model 1	13.57	16510.33	16538.10	0.76	3.85	9.71
trained model 2	18.50	50591.95	50679.39	0.75	8.70	9.80
trained model 3	33.66	32043.12	32012.82	0.74	12.78	20.88
trained model 4	20.82	47331.80	48260.44	0.79	14.16	6.66

Table 5.2: Stable coins experiment results

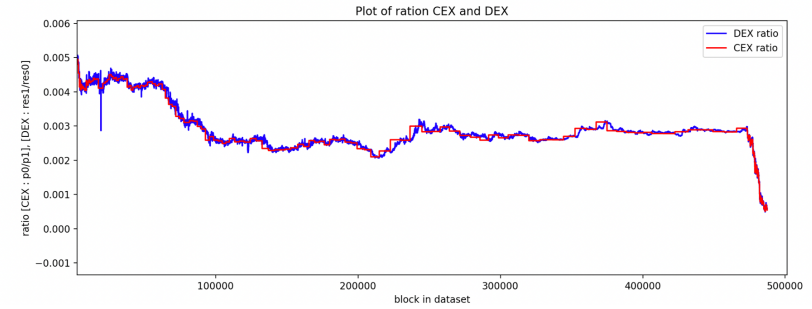
The evaluation of the resulting models involves the use of various metrics. The primary ranking metric used is profit, which is also the reward the agent is optimizing for. Among the trained models, models 3 and 4, as well as the optimized swapping baseline, have achieved the highest profit. The columns indicating the amount of liquidity tokens show the amount remaining in the pool after the testing sequence, which can be used to analyze potential risks taken. The ratio of c0/c1 indicates the coin ratio in the personal wallet, with a ratio over 1 indicating more c0 than c1, and vice versa. For all models allowed to swap, the ratio is below one. The reward decomposition, showing the reward from liquidity provision and the reward due to swapping, helps to understand the strategy used by the agent to generate profit. The reward decomposition provides a useful means of comparing the reward generated by the model with those of the baselines, which either involve adding liquidity or solely swapping. The model's reward value for the swapping strategy can be seen to approach the optimized or front-running swapping baseline value. A similar observation can be made for the liquidity provision strategy. Although the rewards for both the swap and liquidity provision strategies are marginally lower than their respective baselines, the model's overall performance surpasses traditional trading strategies when these two strategies are combined.

5.5 Unstable Coin Experiment

In this section, we will explore additional experiments conducted on a different coin pair consisting of Ether, an unstable coin, and the stablecoin USDC. Due to the inherent volatility of unstable coins, coin pairs including them exhibit greater variation in centralized market prices and more significant movements in liquidity pool reserves.



(a) Stable coin pair



(b) Unstable coin pair

Figure 5.3: Ratio CEX vs Ratio DEX

5.5.1 Hyperparameter Choice

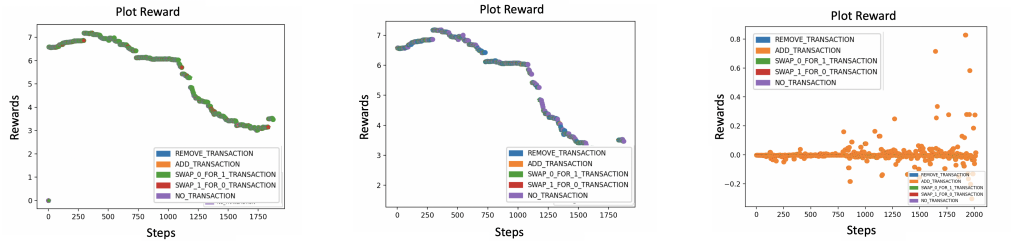
Due to the greater variation in the ratio of coins in the unstable dataset, the hyperparameter choices must be adjusted accordingly. The longer ranges of imbalance could potentially cause the agent to overfit behaviors that are effective only for specific imbalances. To address this issue, we increased the number of environments to 20, each starting the simulation at a different index, which helps to stabilize the training process. Initially, we considered using both the padding and attention architectures, but the padding architecture yielded poor results and was therefore excluded. The selection of hyperparameters was performed in a similar manner as described in Section 4.7.3 and can be viewed in this table A.3 .

5.5.2 Learning Pipeline

During the training process for the unstable coin experiment, the model experienced negative average rewards for an extended period. It was only after training the model for five times longer than the stablecoin model (3,000,000 steps for the stable-coin experiments, 15,000,000 steps for unstable coin experiments) that noticeable performance improvement was observed. This could be attributed to the greater diversity of the data, which requires the agent to be exposed to more data before developing a reliable trading strategy.The significance of the start-index frequency in the unstable coins experiments is the same as in the stable coins experiments 5.4.2.

5.5.3 Baselines

The baselines used for performance assessment are the same as for the stablecoin experiments 5.4.3.



(a) Front-running (b) Optimized (c) Liquidity Provision

Figure 5.4: Computed Baselines: (a) Result of front-running strategy, (b) Result of optimized swapping strategy (c) Results of liquidity provider strategy

5.5.4 Experiment and Results

Model	Reward Elements	Normalization	Network Architecture	Training steps	Start-index Frequency	Attention parameters
trained model 1	short + long (LP)	yes	padding	10'000'000	512	-
trained model 2	short + long (LP)	yes	attention	15'000'000	512	nbr heads = 4
trained model 3	short + long (LP)	yes	attention + gumble	15'000'000	512	nbr heads = 4
trained model 4	short + long (LP)	yes	attention + gumble	15'000'000	512	nbr heads = 12
trained model 4	short + long (LP)	yes	attention + gumble	15'000'000	16	nbr heads = 12

Table 5.3: Overview unstable coin experiments

The hyperparameters for the models were optimized using grid search, while some parameters were carried over from previous implementations due to their proven good performance. The values of these hyperparameters are listed in Table A.3.

Upon initial experimentation **(Trained Model 1)**, we observed that the padding network architecture failed to perform adequately. It appears that the increased frequency of pending transactions per block introduced a high degree of variance in the zero and non-zero values presented to the network. Consequently, we discontinued the use of this approach and focused solely on the attention network architecture for subsequent experiments.

Following the rejection of the padding network approach, we opted to reinstate the attention model **(Trained Model 2)**. This model's filtering technique ensures that the neural network always receives a consistent number of matched transactions, resulting in a reduced variance in the information content. However, the training performance of this model improves at a slower rate compared to the model used for stablecoins, taking at least 10 million steps before reaching a positive reward. Additionally, the process is highly sensitive to the randomness of the starting indexes of the dataset, which can have a substantial impact on the model's overall performance. Two models having the same parameters can exhibit completely different learning behavior. The outcome is not optimal, as it leads to negative reward outputs, inadequate exploitation of imbalances, and a lack of positioning.

In the standard multi-head attention implementation, the softmax function normalizes attention scores, which are then utilized to compute a weighted sum of input values. During this process, input values from all transactions are implicitly combined to a certain extent. For instance, given three transactions with attention scores of 0.9, 0.1, and 0.1, the final output remains influenced by all three transactions, even though the first transaction has a substantially higher attention score. This could be a factor contributing to the stagnation of our RL agent's performance at a lower level. Gumbel Softmax [55] serves as an alternative approach that can offer a clearer separation between different transactions. It is a continuous relaxation of the discrete argmax function, enabling it to approximate a hard (i.e., one-hot) assignment of attention scores. Consequently, Gumbel Softmax can generate attention scores closer to binary values (e.g., [1, 0, 0]), leading to a more distinct selection of a single transaction rather than blending values from multiple transactions. In the subsequent experiment **(Trained Model 3)**, the softmax function from the basic attention implementation was replaced with Gumbel Softmax.

As part of an additional experiment **(Trained Model 4)**, the multihead-attention layer of the network was modified to increase the number of attention heads. This adjustment allows the model to capture more nuanced relationships between input tokens, providing a higher degree of granularity. Furthermore, the increased number of attention heads is anticipated to improve the model's generalization capacity, which is particularly vital for managing the diverse dataset encountered in the unstable coin implementation. The performance of the model improved slightly when increasing the number of heads from 4 to 12. However,

upon a higher increase (20 heads), the performance dropped significantly.

In a final experiment, referred to as **(Trained Model 5)**, the reset value was substantially decreased from 512 to 16, while other parameters and hyperparameters from the previous model were retained. This was done as the motivation for liquidity provision was proven not to be adopted by prior models and therefore the long reset ranges were not needed anymore and This change was driven by the hypothesis that having 20 environments with different starting indices might not offer sufficient diversity for the model to further enhance its performance. As a result, the model's performance improved to the extent that it now matches the front-running baseline.

The average metrics across all testing sets are presented in the table below 5.4. Furthermore, a visualization of the resulting agent behavior for various models can be found here: A.2. This visualization displays the overall reward distribution and provides a closer look at the agent's actions concerning pending transactions.

Model	Profit	Amount uniswap c0	Amount uniswap c1	Ratio c0/c1	Reward from LP [usd]	Reward from swap [usd]
lower bound random	-76.31708	1052.60	1054.49	1.01	0.29	-76.61
baseline liquidity provider	0.8162	50003.24	145.40	1.00	0.8162	0.0
baseline front-running	592.66	0.0	0.0	0.49	0.0	592.66
baseline optimized swapping	594.35	0.0	0.0	0.49	0.0	594.35
trained model 2	567.91	1633.77	4.60	0.56	0.045	567.86
trained model 3	577.72	0.0	0.0	0.56	0.0	577.72
trained model 4	587.67	3530.12	9.99	0.55	0.043	587.62
trained model 5	592.38	215.98	0.60	0.51	0.002	592.378

Table 5.4: Unstable coin experiments results

The outcomes of the padding network experiment will not be incorporated into subsequent analyses, as the performance failed to achieve positive reward values during training. The evaluation metrics for the other models are identical to those employed in the stable coin implementation; thus, their rationale will not be reiterated here. Notably, the trained models do not outperform the optimized baselines, but the performance of the last model **(Trained Model 5)** is comparable with the one of the front-running baseline. In contrast to the stable coin experiments, the profitability of liquidity provision is considerably lower, contributing only a minimal portion of the total profit. In terms of the cryptocurrency ratios held in the personal wallet following testing, significant deviations from 1 are observed for both the agent and the swapping baselines (optimized and front-running), indicating a substantial loss of portfolio diversification. To provide further insights, we will examine the agent's behavior in greater detail within the discussion section 6.2.2.

Chapter 6

Discussion

In the discussion of the results, a comprehensive analysis of various model outputs will be conducted, which will involve comparing them with each other and the established baselines. The discussion will focus on evaluating performance as well as assessing risk factors. We will begin by promptly examining the initial results obtained from the simulated data, followed by a comprehensive analysis of the historical data results.

6.1 Simulated Data Environment

Since the simulated data experiments served as a preliminary step before moving on to real data experiments, their discussion will be brief.

6.1.1 Performance Discrete Model

The discrete model performs well for a limited number of trading transactions. However, as the number of transactions increases and their variability grows, this approach becomes impractical. The main reason is that with more transactions, the number of possible positions grows exponentially, which cannot be accommodated by a fixed discrete action space. Furthermore, since the action space must be predetermined, it cannot be adjusted dynamically to accommodate varying transaction numbers.

6.1.2 Performance Continuous Model

The continuous model successfully addresses the issue of the increasing and fluctuating action space encountered in the discrete model, while achieving comparable performance results. By using a continuous gas price value, managing positioning with a larger or varying number of transactions becomes simpler as only one value is needed. However, it is important to note that the experiment in this

section solely focused on modeling the gas price and did not include modeling the amount or the agent's liquidity provider role.

6.2 Historical Data Environment

The performance of the agent in the historical data setting can be more thoroughly assessed especially with the help of performance baselines. Additionally, the aspect of risk will be looked at.

6.2.1 Stable coins

Performance and Risk

The assessment of performance and risk will mainly concentrate on the top-performing model, referred to as **(Trained Model 3)**, while also including insights on **(Trained Model 4)** within the risk analysis.

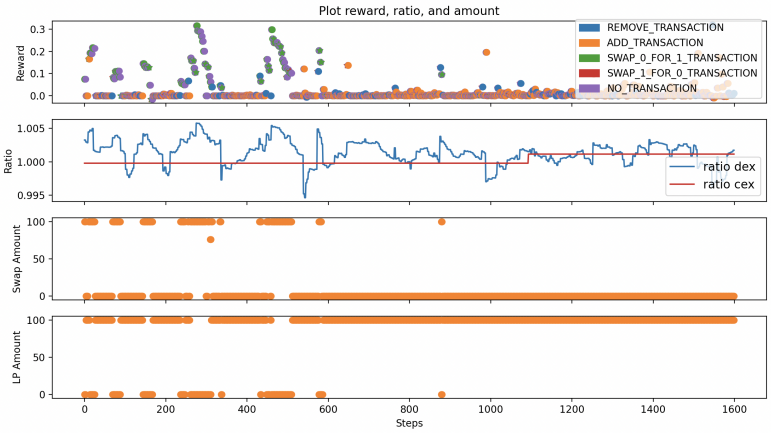
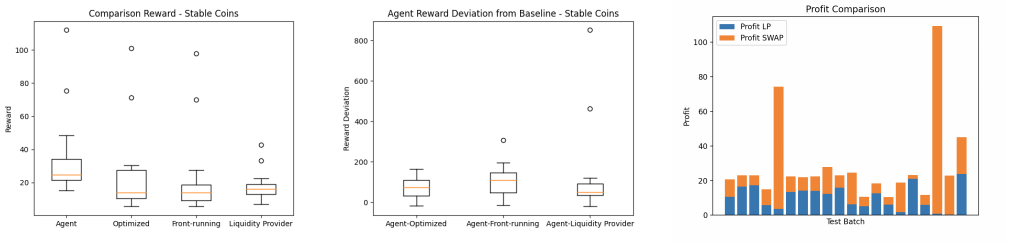


Figure 6.1: Plot of Reward, Ratio dex ($\frac{\text{reserveC1}}{\text{reserveC0}}$), Ratio cex ($\frac{\text{PriceC0}}{\text{PriceC1}}$), and Transaction Amount (USD) for the LP actions and the Swap actions

By examining the agent's behavior in relation to the centralized (cex) and decentralized (dex) ratios, and considering the amount invested by the agent, we can observe that the agent's actions and therefore its rewards are closely aligned with fluctuations in the ratios. The agent effectively executes swaps when significant imbalances arise, utilizing the maximum permitted amount for such actions and provides liquidity when the price discrepancies are smaller.

After comparing the agent's performance to various baselines, it is evident that the agent outperforms all of them by utilizing a mixed strategy. This strategy involves both swapping with a pattern similar to that of the front-running or

optimized baselines and providing liquidity simultaneously. Furthermore, the agent's swapping performance is superior to that of the front-running baselines. This is because the agent has access to information about pending transactions and does optimize its positioning accordingly. By leveraging this information, the agent is able to make more informed decisions about its positioning, resulting in better performance.

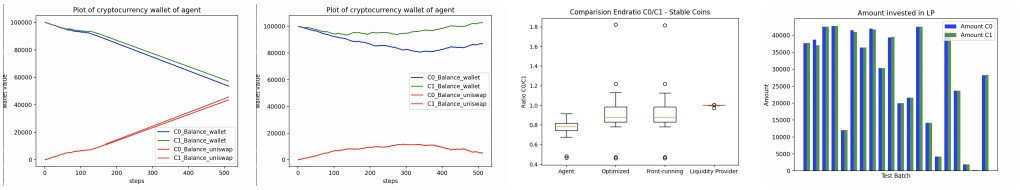


(a) Reward (b) Reward Deviation (c) Profit Decomposition

Figure 6.2: (a) shows the summed reward distribution overall test set for agent and baselines, (b) shows the reward deviation of our agent regarding each baseline over the whole test set, (c)shows the agents profit decomposition for each test set into profit generated by the liquidity provision and the one from swapping

The optimized baseline outperforms other baselines in terms of profit, with a higher number of high-profit pinnacles. However, it also exhibits the greatest variability in profit distribution over the testing set, highlighting the instability of this trading strategy. Conversely, the liquidity provider baseline yields lower step-wise profit but a more stable accumulation of profit over time, resulting in a moderate overall profit sum. This baseline also has the narrowest range of profit distribution across different testing intervals, indicating a more consistent performance. Analyzing the agent's profit plot, we can observe that its distribution of profit across the testing set falls between that of the optimized and liquidity provider baselines. This suggests that the agent adopts a strategy that balances the stability of the liquidity provision approach with the volatility of the optimized swapping baseline, resulting in a profitable yet stable trading approach. The observation is strengthened by examining both the agent's reward deviation and the breakdown of its reward into swapping and liquidity provision components. The reward obtained from liquidity provision is more stable, providing a foundation with lower variance. Conversely, the reward obtained from swapping exhibits greater variability across different testing sets, despite yielding higher returns overall.

In order to evaluate risk, we will differentiate between two types of risks: the risk associated with the agent's personal wallet and the risk associated with providing liquidity in the pool 3.5. The agent's strategy depends on the underlying data, and thus the mix of swapping or adding will be adapted to the specific



(a) LP Risk (b) Wallet Risk (c) Wallet Ratio (d) LP Amount

Figure 6.3: (a) shows a test batch where the risk dominating is the liquidity provision risk, (b) Shows a test batch where the wallet risk is prevalent, (c) shows the ratio of coin 0 to coin 1 that the agent possesses after the test trading which can be linked to the wallet risk, and (d) shows the amount of coins in the liquidity pool which can be linked to the liquidity provision risk.

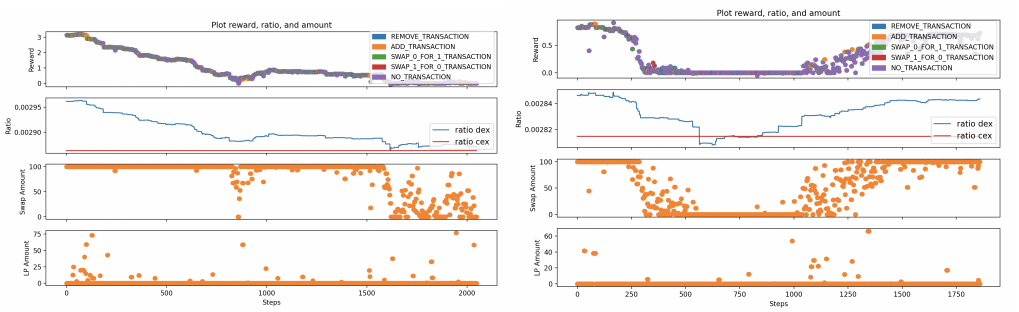
situation. In Figure 6.3a, the dominant strategy is providing liquidity, which can be observed by the increasing liquidity token balances shown in the red line. Therefore, this test batch is more strongly affected by liquidity provider risk. In contrast, in Figure 6.3b, the increase in the liquidity pool is less prevalent, but the gap between the amount of coin 0 and coin 1 becomes larger, indicating a higher level of personal wallet risk for this test batch. If we analyze the wallet ratio of our agent over the entire testing set 6.3c, we can observe that the ratio is consistently below 1, indicating that the agent frequently trades coin 0 for coin 1. Similar behavior can be observed in the front-running/optimized baselines, although their imbalance is lower than that of our agent. Assuming that the price movements of each cryptocurrency are independent, this suggests that the agent has a higher risk due to a less diversified cryptocurrency portfolio. To analyze the risk associated with the liquidity pool asset, we can plot the amount of coins the agent has in the liquidity pool 6.3d. The agent faces the risk of liquidity slippage, which can affect the amount of coins they provided into the liquidity pool and, therefore, can be viewed as the agent's value at risk.

In the experiment involving long-term rewards based on personal wallet imbalances (**(Trained Model 4)**, an interesting pattern emerges. The ratio of wallet coins gravitates marginally closer to one in comparison to all other trained models. The average profit derived from liquidity provision is higher than the model without risk embedding, while the profit from swaps decreases substantially, leading to a reduction in overall profit. This behavior could be attributed to the long-term rewards that penalize excessive swaps in a single direction. Despite the relatively minor reduction in imbalance compared to the decline in profits, this experiment provides important insights into the potential of incorporating tailored rewards to guide agent behavior toward a particular objective. This area of research could be more thoroughly explored in future experiments.

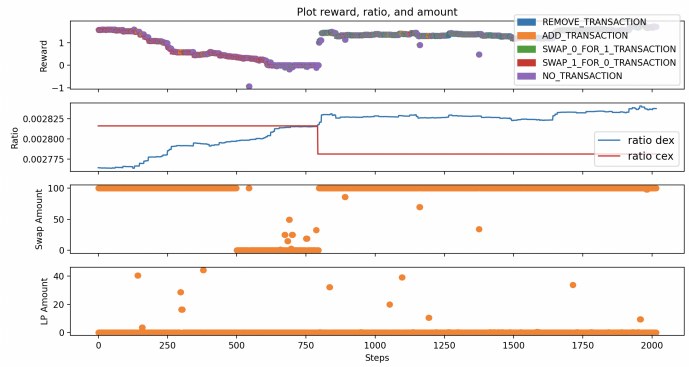
6.2.2 Unstable coins

The assessment of performance and risk will mainly concentrate on the most successful model, which utilizes an attention layer network, the Gumbel-softmax, and 12 heads and the start-index frequency of 16 steps (referred to as **Trained Model 5**). Due to its inability to achieve satisfactory performance, the model employing the padding network will not be discussed further in this evaluation.

Performance and Risk



(a) (b)



(c)

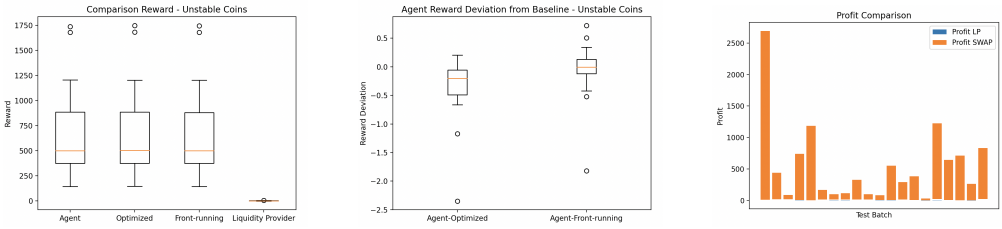
Figure 6.4: Plot of Reward, Ratio dex ($\frac{\text{reserveC1}}{\text{reserveC0}}$), Ratio cex ($\frac{\text{PriceC0}}{\text{PriceC1}}$), and Transaction Amount (USD) for the LP actions and the Swap action (on three different testing sets)

Upon evaluating the agent's performance across three distinct testing sets, we can assess its effectiveness and pinpoint both the aspects contributing to optimal performance and the shortcomings. In the plot 6.4a, it is evident that the agent capitalizes on the substantial price disparity between the cex and dex markets. The reward structure and investment amounts closely align with the differences

in market ratios. In plot 6.4b, the liquidity pool reserve ratio exhibits a decline around the middle of the testing set, with a minor ratio crossing, before rising again. The agent reacts to the price discrepancy as it approaches the crossing point, choosing to cease investments in swaps and withhold instead. This decrease in investment amounts aligns with the ratio plot, indicating optimal behavior. Nevertheless, the short interval where ratio CEX is larger than the DEX ratio is not capitalized upon. The results from the third testing set, as seen in 6.4c, present a scenario where an abrupt decrease in the CEX ratio occurs. The agent deals well with the crossing itself but gets a reward below zero when the ratios are in close proximity to one another.

This challenge can be partially understood by examining the fluctuations between CEX and DEX ratios, as demonstrated in Figure 5.3.For the stable coin implementation, the CEX and DEX ratios experienced fewer significant fluctuations in overall value but had a higher frequency of crossing points. The increased occurrence of ratio crossings in the training dataset might have contributed to the stable model's superior ability to learn appropriate responses in these situations. For future experiments, an additional training dataset could be constructed to focus on the condensed ratio crossings.

The second limitation relates to the agent's struggle to learn optimal positioning strategies A.2. This issue may be due to the relatively small increase in reward when comparing the optimized baseline with the front-running baseline. The incremental reward obtained by the agent for using gas price values to determine positioning may be so subtle at each step that the policy update process fails to adequately capture these changes. As a result, the agent has difficulty understanding the significance of positioning based on gas price values.



(a) Reward (b) Reward Deviation (c) Profit Decomposition

Figure 6.5: (a) Summed reward distribution for agent and baselines across the test set, (b) reward deviation of the agent with respect to each baseline over the entire test set, and (c) profit decomposition for each test set, separating profits from liquidity provision and swapping.

As depicted in Figure 6.5, the visual representations provide evidence that the agent's performance aligns with the front-running agent, as indicated by a reward deviation approaching zero, with occasional deviations represented by outliers.

The comparison between the two swapping baselines in terms of reward distribution and deviations from the agent supports the earlier statement regarding the small increase in profit when utilizing positioning. On the other hand, the liquidity provider baseline is significantly lower compared to the other baselines, as demonstrated in the reward distribution and reflected in the profit decomposition of the agent. This explains partly why the agent cannot outperform the baselines using a simple mixed strategy

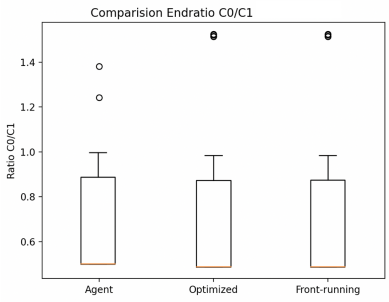


Figure 6.6: Wallet Ratio Unstable

The risk analysis will concentrate exclusively on the risk associated with the agent's personal wallet, given that the trained policy results in a negligible amount of liquidity provision. Both the model and the baselines exhibit imbalances in the end ratio of the personal wallet. Interestingly, the optimized and front-running baselines demonstrate a slightly more pronounced imbalance in the personal wallet over the testing set, leading to a higher risk exposure.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

In this thesis, we demonstrated that a reinforcement learning (RL) agent can achieve positive trading results on both stable and unstable coin pairs. For the stablecoin implementation, a mixed strategy of swap and liquidity provision was employed, resulting in higher performance compared to baseline approaches. However, in the case of the unstable coin implementation, stronger imbalances made liquidity provision more challenging, thus upper-bounding the agent's performance to an optimized swapping strategy.

We gained insights into constructing an RL agent for trading purposes and experimented with various hyperparameters to observe their influence on the agent's behavior. A multi-stage training pipeline was developed that includes multiple entropy coefficient resets and penalization of negative rewards to enhance performance. An approach for designing the action space was devised to accommodate actions requiring both discrete and continuous elements, using an encoding scheme. Regarding the observation space, experiments were conducted with variable-length observations employing different techniques. Initially, a padding network was used to handle a smaller variance in the number of observations, which was later replaced with a more sophisticated attention layer approach. Additionally, performance improvement was achieved by replacing the softmax layer with a Gumbel-softmax layer, which helps in avoiding information loss.

Additionally, we showcased that customized reward structures can be integrated into the training process, fostering desired behavior. By emphasizing liquidity provision within the reward structure, the agent can be encouraged to select actions prioritizing this aspect. Furthermore, incorporating risk management into the network using targeted rewards can help the agent learn to minimize risk in its actions, leading to more informed decisions.

The training challenges encountered with the unstable coin pair highlight the complexities in developing an RL agent capable of generalizing across various coin types, as evidenced by the contrasting performance on the two coin pairs.

This study emphasizes the potential of reinforcement learning (RL) in cryptocurrency trading and offers valuable insights into effective training strategies for RL agents within this domain. Although no outstanding strategies were discovered, crucial design decisions were identified, showcasing how to effectively train an RL agent to navigate the intricate landscape of cryptocurrency trading.

7.2 Outlook

The performance analysis of the unstable model suggests that the agent faces challenges in achieving optimal performance when the price discrepancies between dex and cex market prices approach crossing points. As mentioned earlier, this behavior could stem from the dataset lacking sufficient crossings, resulting in the agent's inability to optimize its policy. A possible future approach might include developing a training pipeline that emphasizes a higher frequency of data containing crossings during the training process.

The simulation developed in this thesis presents an opportunity to expand the investigation of trading strategies to a wider range of coin pairs beyond those studied in this research. Notably, the approach of combining two unstable coin pairs has not yet been investigated and represents an area of potential exploration.

Another potential area of exploration is to enhance the versatility of the RL agent by allowing it to trade on multiple liquidity pools simultaneously. This can be achieved by scaling up the simulation environment to enable multipool trading. By learning diverse strategies across various markets, the agent can improve its performance and robustness in different market conditions.

Furthermore, there is potential for the research on custom long-term rewards to be expanded upon in order to integrate more complex trading preferences into the agent's behaviour.

Bibliography

[1] "Uniswap v1 smart contracts," Available at: https://docs.uniswap.org/contracts/v1/overview, accessed: [12.12.22].

[2] DefiLlama, "DEXs on BSC - defillama," https://defillama.com/dexs, accessed: April 23, 2023.

[3] B. F. K. Y. R. Z. Deng, Y. and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," 2017.

[4] L. Z. L. X. Wen, Y. and J. Wang, "Ensemble learning based reinforcement learning for stock trading strategy," 2021.

[5] Y. Zhang and Y. Zhou, "Multi-objective reinforcement learning-based trading strategy optimization for cryptocurrencies," 2021.

[6] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014.

[7] B. V. et al, "A next-generation smart contract and decentralized application platform." 2014.

[8] L. F. G.-G. C. e. a. Khan, S.N., "Blockchain smart contracts: Applications, challenges, and future trends," 2021.

[9] J. Zakrzewski, "Towards verification of ethereum smart contracts: a formalization of core of solidity," 2019.

[10] D. Meijer, "Defi and regulation: the european approach," 2021.

[11] Y. Wang, Y. Chen, H. Wu, L. Zhou, S. Deng, and R. Wattenhofer, "Cyclic arbitrage in decentralized exchanges," 2022.

[12] "Uniswap interface v2," https://v2.info.uniswap.org/, accessed: April 21, 2023.

[13] N. Z. Hayden Adams and D. Robinson, "Uniswap v2 core." 2020.

[14] "Front running," https://www.nasdaq.com/glossary/f/front-running.

[15] J. Xu and B. Livshits, "The anatomy of a cryptocurrency pump-and-dump scheme." 2019.

[16] O. G. B. P. Forum, "Blockchain transparency report," 2019.

[17] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," 2020.

[18] S. Byrne, "An exploration of novel trading and arbitrage methods within decentralised finance," 2021.

[19] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," 2019.

[20] N. Boonpeam, W. Werapun, and T. Karode, "The arbitrage system on decentralized exchanges," in *2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2021.

[21] M. A. Samsuden, N. M. Diah, and N. A. Rahman, "A review paper on implementing reinforcement learning technique in optimising games performance," in *2019 IEEE 9th International Conference on System Engineering and Technology (ICSET)*, 2019.

[22] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," 2013.

[23] Y. Zeng, J. Sun, X. Li, X. Gao, X. Yang, and J. Ren, "Deep reinforcement learning for supply chain optimization: A review," 2020.

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," 2016.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2018.

[26] B. H. Rui Nian, Jinfeng Liu, "A review on reinforcement learning: Introduction and applications in industrial process control," 2020.

[27] P. Wei, S. Chen, J. Zheng, B. Yang, and Y. Zhou, "Graph reinforcement learning for portfolio management," 2021.

[28] X. Gao, W. Chen, and W. Liu, "Reinforcement learning for asset management," 2021.

[29] X. Cao, Y. Luo, W. Chen, and S. Kou, "Reinforcement learning for option pricing, hedging, and trading," 2020.

[30] G. Huang, X. Zhou, and Q. Song, "Deep reinforcement learning for portfolio management," 2022.

[31] G. D. M. A. Y. M. P. K. Jagdish Bhagwan Chakole, Mugdha S. Kolhe, "A q-learning agent for automated trading in equity stock markets," 2021.

[32] L. Zhao, H. Liu, Y. Xu, and J. Wu, "Deep reinforcement learning for option pricing and risk management," 2021.

[33] S. Quinteiro dos Santos, C. Chukwuocha, S. Kamali, and R. Thulasiram, "An efficient miner strategy for selecting cryptocurrency transactions," 2019.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[35] V. Konda and J. Tsitsiklis, in *Advances in Neural Information Processing Systems*, 1999.

[36] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2015.

[37] A. Aigner and G. Dhaliwal, "Uniswap: Impermanent loss and risk profile of a liquidity provider," 2021.

[38] G. V. Nartea and Y. Wu, "The diversification benefits of cryptocurrencies: Evidence from bitcoin and ethereum," 2021.

[39] M. F. L. Siti Nur Iqmal Ibrahim, Masnita Misiran, "Geometric fractional brownian motion model for commodity market simulation," 2021.

[40] T. Szabados, "An elementary introduction to the wiener process and stochastic integrals," 2010.

[41] Z. Ding, T. Yu, Y. Huang, H. Zhang, G. Li, Q. Guo, L. Mai, and H. Dong, "Efficient reinforcement learning development with rlzoo," 2021.

[42] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters in on-policy reinforcement learning? a large-scale empirical study," 2020.

[43] M. Kiran and M. Ozyildirim, "Hyperparameter tuning for deep reinforcement learning applications," 2022.

[44] F. Gogianu, T. Berariu, M. Rosca, C. Clopath, L. Busoniu, and R. Pascanu, "Spectral normalisation for deep reinforcement learning: an optimisation perspective," 2021.

[45] A. Raffin, J. Kober, and F. Stulp, "Smooth exploration for robotic reinforcement learning," 2021.

[46] F. Gao and H. Zhong, "Study on the large batch size training of neural networks based on the second order gradient," 2020.

[47] L. Zhang, T. Wu, S. Lahrichi, C.-G. Salas-Flores, and J. Li, "A data science pipeline for algorithmic trading: A comparative study of applications for finance and cryptoeconomics," 2022.

[48] E. J. Elton, M. J. Gruber, and C. R. Blake, "Survivorship bias and mutual fund performance," 1996.

[49] B. F. Arnaud Laurent, Luce Brotcorne, "Transaction fees optimization in the ethereum blockchain," 2022.

[50] Gürcan, A. Ranchal-Pedrosa, and S. Tucci-Piergiovanni, "On cancellation of transactions in bitcoin-like blockchains," 2018.

[51] YCharts, "Ethereum average gas price," https://ycharts.com/indicators/ethereum_average_gas_price, 2023, accessed on April 21, 2023.

[52] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[53] Z. Cai, A. Ravichandran, S. Maji, C. Fowlkes, Z. Tu, and S. Soatto, "Exponential moving average normalization for self-supervised and semi-supervised learning," 2021.

[54] A. Bhatt, M. Argus, A. Amiranashvili, and T. Brox, "Crossnorm: Normalization for off-policy td reinforcement learning," 2019.

[55] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," 2017.

Appendix A

Additonal Data

Table A.1: Hyperparameters for the discrete and continuous models

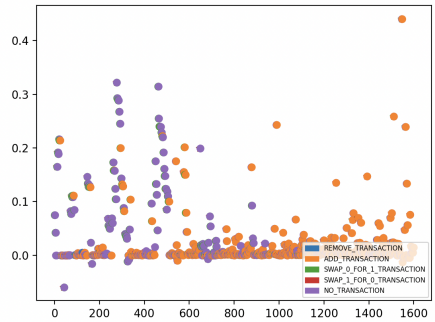
Hyperparameter	**Discrete Model**	**Continuous Model**
activation_fn	th.nn.Tanh	th.nn.Tanh
net_arch	[vf=[64, 64], pi=[64, 64])]	[(vf=[64, 64, 64], pi=[64, 64, 64])]
batch_size	128	128
n_steps	1024	1024
gamma	0.99	0.9
learning_rate	exp. decay (1e-3, 0.9)	exp.decay (0.0012, 0.9)
ent_coef	0.03	0.003
clip_range	0.25	0.4
n_epochs	10	10
gae_lambda	0.9	0.8
max_grad_norm	5	0.6
vf_coef	0.5	0.8
use_sde	-	True
normalization	-	Moving average

Table A.2: Hyperparameters for stable models

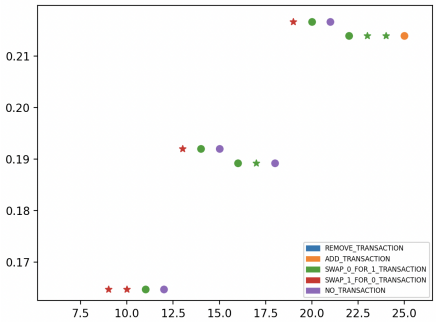
Hyperparameter	**Model 1**	**Model 2**	**Model 3, 4**
activation_fn	th.nn.Tanh	th.nn.Tanh	th.nn.Tanh
net_arch	[128, 128, 128]	[128, 128, 128]	[128, 128, 128]
batch_size	128	64	256
n_steps	2048	2048	2048
LR - exp.decay(start, decay)	0.01 ,0.9	0.002, 0.9	0.002, 0.9
ent_coef - exp.decay(start, decay)	0.003, 0.9	0.003, 0.9	0.003, 0.9
clip_range	0.2	0.4	0.4
n_epochs	10	5	10
gae_lambda	0.8	0.92	0.92
max_grad_norm	1	0.9	0.9
vf_coef	0.8	0.63	0.75
use_sde	True	True	True
normalization	Moving average	Moving average	Moving average

Table A.3: Hyperparameters for unstable models

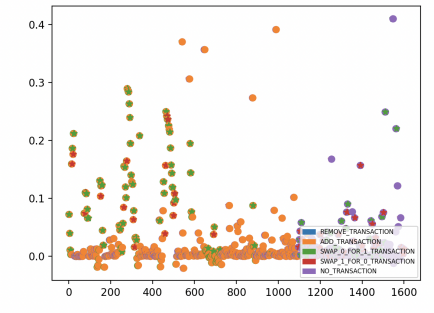
Hyperparameter	**Model 1**	**Model 2, 3, 4, 5**
activation_fn	th.nn.Tanh	th.nn.Tanh
net_arch	[128, 128, 128]	[128, 128, 128]
batch_size	256	256
n_steps	2048	2048
LR - exp.decay(start, decay)	0.01 ,0.9	0.1, 0.9
ent_coef - exp.decay(start, decay)	0.003, 0.9	0.003, 0.9
clip_range	0.3	0.3
n_epochs	10	15
gae_lambda	0.89	0.92
max_grad_norm	1	0.8
vf_coef	0.82	0.71
use_sde	True	True
normalization	Moving average	Moving average



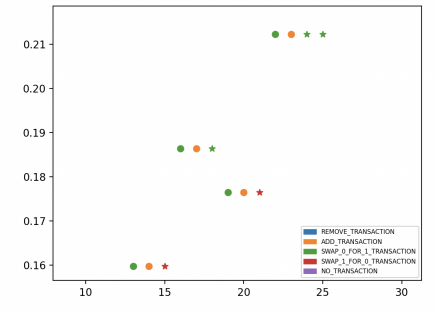
(a) Trained Model 1 - Full



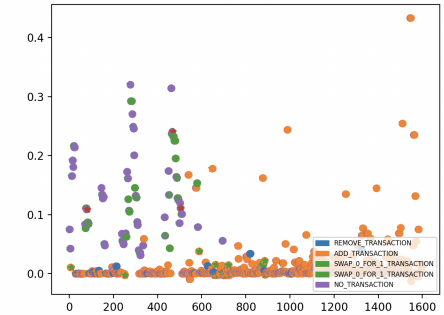
(b) Trained Model 1 - Close



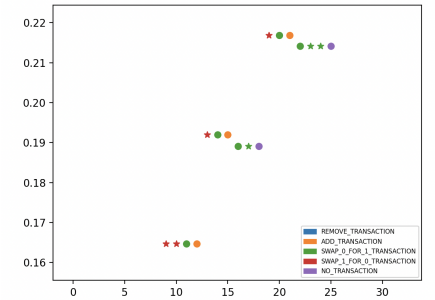
(c) Trained Model 2 - Full



(d) Trained Model 2 - Close

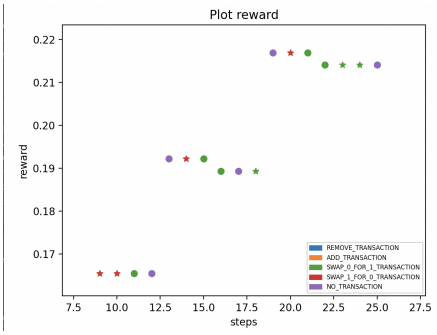


(e) Trained Model 3 - Full



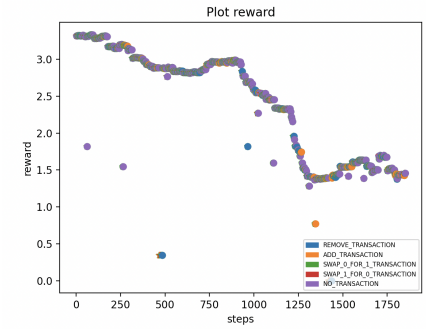
(f) Trained Model 3 - Close

(g) Trained model 4 - Full

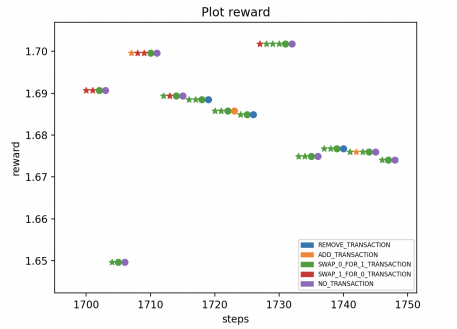


(h) Trained Model 4 - Close

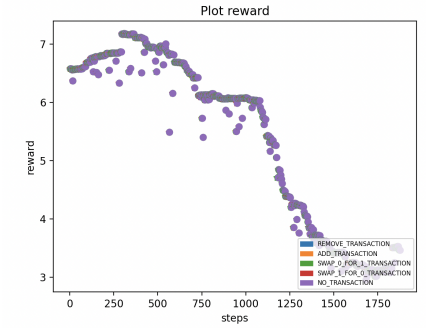
Figure A.1: Stable coins: The images denoted on the left show the full reward output of the specific model, and the images on the right side show a close-up of the result where the positioning is shown. (x-axis = rewards, y-axis=steps)



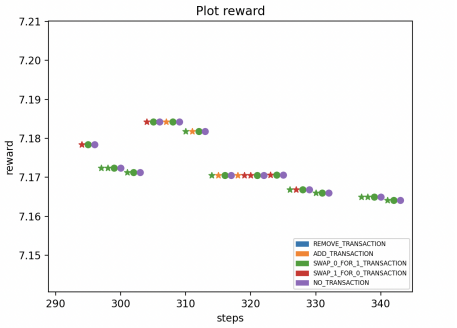
(a) Trained Model 2 - Full



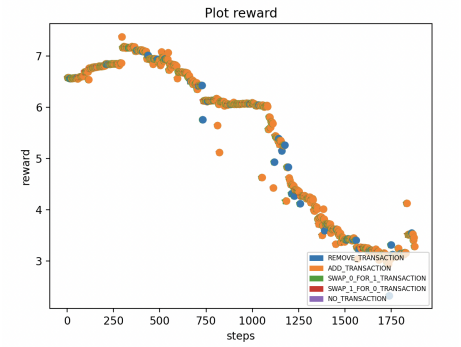
(b) Trained Model 2 - Close



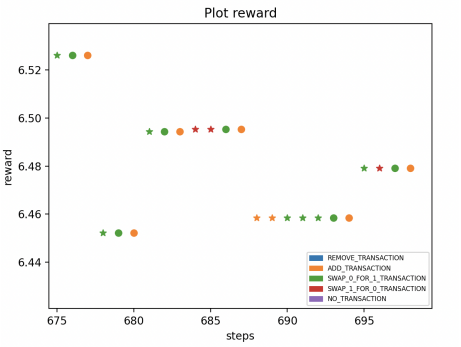
(c) Trained Model 3 - Full



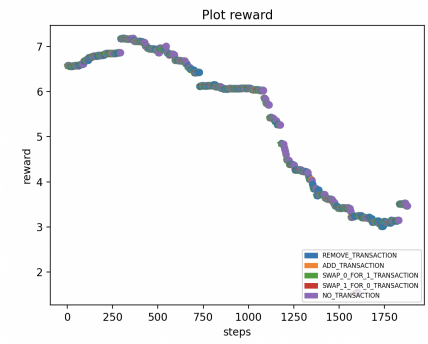
(d) Trained Model 3 - Close



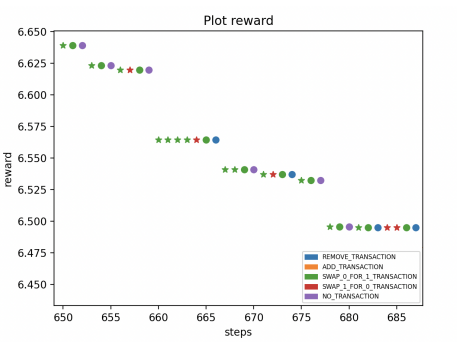
(e) Trained Model 4 - Full



(f) Trained Model 4 - Close



(g) Trained Model 5 - Full



(h) Trained Model 5 - Close

Figure A.2: Unstable coins: The images denoted on the left show the full reward output of the specific model, and the images on the right side show a close-up of the result where the positioning is shown. (x-axis = rewards, y-axis=steps)