Deep reinforcement learning for hedging concentrated liquidity positions

Oskari Möykkynen

School of Science

Espoo 11.12.2025

Supervisor

Prof. Ahti Salo

Advisor

Prof. Ahti Salo

Aalto University
School of Science

Aalto University
School of Science

Author Oskari Möykkynen

Title Deep reinforcement learning for hedging concentrated liquidity positions

Degree programme Mathematics and Operations Research

Major Systems and Operations Research

Supervisor Prof. Ahti Salo

Advisor Prof. Ahti Salo

Date 11.12.2025 **Number of pages** 74 **Language** English

Abstract

Decentralized finance has introduced new forms of market making through automated market makers, where users provide liquidity to decentralized exchanges such as Uniswap. In Uniswap v3, liquidity providers (LPs) can concentrate their liquidity within custom price ranges, improving capital efficiency but introducing exposure to impermanent loss and nonlinear portfolio risk. Managing this exposure dynamically poses a challenge as the LP position's token composition changes continuously with market prices.

This thesis investigates whether a Deep Reinforcement Learning (DRL) agent can effectively hedge a Uniswap v3 LP position using cryptocurrency futures. A simulated Uniswap environment is developed to model concentrated liquidity providing positions and the hedge positions tied to it. The hedging problem is framed as a sequential decision process, where the agent seeks to minimize downside portfolio variance while maintaining upside exposure. The Proximal Policy Optimization algorithm is applied to learn an adaptive hedging policy, which is evaluated against baseline strategies such as fixed-frequency rebalancing.

The results show that the DRL-based hedges can outperform the baseline strategies, achieving higher average portfolio returns, with similar average drawdowns. However, the learned policies varied between agents, and although they reduced downside variance in many cases, they were also more likely to experience larger maximum drawdowns. These findings indicate that reinforcement learning offers a promising but complex approach for managing the nonlinear risks of concentrated liquidity provision in decentralized exchanges.

Keywords deep reinforcement learning, hedging, concentrated liquidity, uniswap

Aalto-yliopisto
Perustieteiden
korkeakoulu

Tekijä Oskari Möykkynen

Työn nimi Syvävahvistusoppiminen keskitettyjen likviditeettipositioiden suojaamisessa

Koulutusohjelma Mathematics and Operations Research

Pääaine Systems and Operations Research

Työn valvoja Prof. Ahti Salo

Työn ohjaaja Prof. Ahti Salo

Päivämäärä 11.12.2025 **Sivumäärä** 74 **Kieli** englanti

Tiivistelmä

Hajautetut rahoitusjärjestelmät ovat tuoneet markkinoille uusia markkinatakaamisen muotoja automatisoitujen markkinatakaajien kautta, joissa käyttäjät tarjoavat likviditeettiä hajautetuille pörsseille, kuten Uniswapille. Uniswap v3 -versiossa likviditeetin tarjoajat (LP) voivat keskittää likviditeettinsä valituille hintaväleille, mikä parantaa pääoman tehokkuutta, mutta altistaa samalla väliaikaiselle tappiolle ja epälineaariselle portfolioriskille. Portfolioriskin hallinta dynaamisesti on haastavaa, johtuen LP-sijoituksen virtuaalivaluuttakoostumuksen jatkuvasta muutoksesta markkinahintojen mukana.

Tässä työssä tutkitaan, voiko syvävahvistusoppimiseen (DRL) perustuva agentti tehokkaasti suojata Uniswap v3 -likviditeettiposition virtuaalivaluuttafutuureja hyödyntäen. Työssä kehitetään simuloitu Uniswap-ympäristö, joka mallintaa keskitettyä likviditeetin tarjoamispositiota sekä siihen liittyviä suojauspositioita. Suojausongelma kuvataan sekventiaalisena päätöksentekoprosessina, jossa agentin tavoitteena on minimoida salkun laskusuhdanneriski säilyttäen samalla nousupotentiaali. PPO-algoritmia käytetään mukautuvan suojauspolitiikan oppimiseen, ja sen suorituskykyä verrataan yksinkertaisiin suojaamisstrategioihin.

Tulokset osoittavat, että agentit voivat oppia vertailustrategioita parempia suojautumisstrategioita, kun katsotaan keskimääräistä tuottoa ja keskimääräisiä laskuja. Kuitenkin opitut päätökset vaihtelivat agenttien välillä ja vaikka keskimääräiset laskut olivat vertailustrategioiden kanssa vastaavia, suuremmat yksittäiset laskut olivat tokennäköisempiä agenteille kuin vertailustrategioille. Tulokset viittaavat siihen, että syvävahvistusoppiminen tarjoaa lupaavia mahdollisuuksia hajautettujen pörssien keskitettyjen likviditeettipositioiden riskien hallintaan.

Avainsanat syvävahvistusoppiminen, suojaaminen, keskitetty likviditeetti, uniswap

Contents

Abbreviations

AMM	Automated Market Maker
ANN	Artificial Neural Network
BTC	Bitcoin
CFMM	Constant Function Market Maker
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DeFi	Decentralized Finance
DEX	Decentralized Exchange
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EMA	Exponential Moving Average
ETH	Ethereum
FNN	Feedforward Neural Network
GAE	Generalized Advantage Estimation
IL	Impermanent Loss
KL	Kullback–Leibler (divergence)
LP	Liquidity Provider
LVR	Loss-versus-Rebalancing
MDP	Markov Decision Process
MLP	Multilayer Perceptron
OHLC	Open, High, Low, Close (price data format)
PG	Policy Gradient
PL	Predictable Loss
PnL	Profit and Loss
PPO	Proximal Policy Optimization
RNN	Recurrent Neural Network
RL	Reinforcement Learning
SB3	Stable-Baselines3
SGD	Stochastic Gradient Descent
STD	Standard Deviation
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
USDC	USD Coin
USD	United States Dollar
USDT	Tether (USD-pegged stablecoin)

1 Introduction

Since the inception of Bitcoin in 2008, the cryptocurrency market has seen rapid increase in both value and trading volume. The value of the cryptocurrency market cap reached its peak late 2024 at 3.7 trillion USD, while daily volumes reached over 300 billion USD [1]. One significant driver of this growth is decentralized finance (DeFi), which allows users to borrow, lend and trade without intermediaries. This is clear when looking at the largest decentralized exchange (DEX) Uniswap and its trading volume, which has risen from under 1 billion USD a month in 2020 to at the time of writing around 1 billion USD daily leading to daily fee revenue between 1 to 10 million USD [2].

Uniswap is an Automated Market Maker (AMM), which means instead of relying on a traditional order book to facilitate trading, it relies on liquidity pools [3]. A liquidity pool is a collection of tokens — digital representations of assets such as cryptocurrencies (for example, Ethereum (ETH) or USD Coin (USDC)) — that are locked inside a smart contract. Smart contracts are self-executing computer programs deployed on a blockchain that automatically enforce trading rules and hold assets without intermediaries. Uniswap relies on liquidity providers (LPs) who deposit pairs of tokens into liquidity pools in exchange for trading fees from traders.

More specifically Uniswap is a Constant Function Market Maker (CFMM). Essentially CFMM is a mathematical framework that maintains constant fixed relationship between the token reserves in the liquidity pool at each moment. The most basic version of CFMM is the constant product formula that is used by Uniswap v1 and v2

$$x \times \; y \; = \; k,$$

where, x and y are the reserves, meaning the amounts of each token held in the liquidity pool, and k is a constant. This rule ensures that when one token is bought, its reserve decreases and the other increases, while their product remains constant. As a result, the pool always provides a price for trades and maintains continuous liquidity. To illustrate how a liquidity pool operates, consider an ETH/USDC pool that initially holds 10 ETH and 20,000 USDC, such that the implied price of ETH is $20,000/10 = 2,000$ USDC. When a trader buys 1 ETH from the pool, they must add USDC and remove ETH in a way that keeps the product of reserves constant according to $x \times y = k$. After the trade, the pool might hold 9 ETH and 22,222 USDC, implying a new ETH price of approximately 2,469 USDC. The trade itself causes the price to move because the pool now holds less ETH and more USDC—this price impact reflects the limited liquidity in the pool. In this example, the trader pays 2,222 USDC, which is added to the pool in exchange for receiving 1 ETH. Liquidity providers simultaneously earn a small fee from the transaction for facilitating the trade. Consequently, after the trade, the pool's reserves consist of 9 ETH and 22,222 USDC instead of the 10 ETH and 20,000 USDC held prior to the trade. After the first trade, if another trader sells 1 ETH back to the pool, they would receive 2,222 USDC in return. The pool would then once again hold 10 ETH and 20,000 USDC, satisfying the constant product condition $x \times y = k$.

This constant rebalancing of the pool's token reserves has an important side effect known as impermanent loss (IL). IL is the difference in value between just holding the two tokens and holding the tokens in the liquidity pool. IL arises because the token composition of the liquidity pool, the relative amounts of the two tokens held, changes automatically as traders buy and sell. When the price of one token rises, the pool ends up holding more of the depreciating token and less of the appreciating one. As a result, the total value of the liquidity provider's position can become lower than if the tokens had simply been held outside the pool. IL is referred to as "impermanent" because it can disappear if prices return to their original level, but in practice, it often represents a realized loss once the LP withdraws liquidity.

In traditional AMMs such as Uniswap v2, liquidity is distributed evenly across all possible prices, from zero to infinity. However, in reality, market prices typically fluctuate within a limited range. As a result, a large portion of the capital deposited into the pool remains unused, leading to capital inefficiency and lower returns for liquidity providers.

To address this issue, Uniswap v3 introduced concentrated liquidity, a mechanism that allows LPs to allocate their liquidity within specific price ranges where trading is most likely to occur. Compared to Uniswap v2, this design greatly improves capital efficiency, as liquidity is no longer locked in parts of the price curve that are rarely used. Price curve is the set of all possible prices ranging from zero to infinity. Thus providing liquidity to the full price curve means that equal amount of liquidity is distributed to all prices, but in reality if price is close to zero it most likely wont trade at any large price suddenly, meaning that the liquidity provided in the other end of the price curve remains mostly unused. However, concentrating liquidity also increases exposure to price risk: when the market price moves outside the chosen range, the position stops earning fees and becomes fully exposed to the price movement of a single token. This amplifies impermanent loss, as the token composition of the position can diverge substantially from the initial deposit.

For most LPs, impermanent loss has been shown to exceed the fees earned from trading activity [4]. Therefore, to make liquidity provision worthwhile, LPs require well-designed strategies that balance the potential fee income against the risk of loss. Despite the average LP losing relative to holding the underlying tokens, the substantial amount of fees paid daily, often millions of dollars, continues to attract both participants and academic research. This has led to growing interest in developing models that can optimize liquidity provision and risk management [5, 6, 7, 8].

The constant change in the token composition of liquidity positions introduces the main challenge for risk management. As the composition of an LP position evolves over time, any hedge position must also be dynamically adjusted to maintain coverage. This makes the problem of hedging liquidity positions conceptually similar to option replication [9, 10], where maintaining an exact replica requires continuous trading. Even when transaction costs are small, frequent rebalancing can make hedging expensive. Thus, the key challenge lies in balancing rebalancing costs with tracking performance - minimizing both deviation from the target position and transaction expenses.

This thesis investigates how effectively a Deep Reinforcement Learning (DRL) agent is able to manage the hedge of a Uniswap v3 LP position using cryptocurrency futures. The objective is to evaluate whether DRL can learn an effective dynamic hedging policy that balances the downside protection and upside exposure. The learned policies are compared to baseline hedging strategies by assessing total portfolio performance, including portfolio returns, volatility and drawdowns. The analysis focuses on the hedging of individual LP positions, excluding fee revenue and liquidity allocation optimization.

The thesis is structured as follows. Chapter 2 focuses on the literature on deep reinforcement learning as well as the Proximal Policy Optimization (PPO) algorithm. Chapter 3 presents an overview of liquidity provision as well as the mathematical formulations necessary for the optimizations. Chapter 4 explains the research methods that are used, including the data and models used. The training setup and baseline strategies are presented in Chapter 5, which is followed by results in Chapter 6. Finally, Chapter 7 consists of discussion and conclusions.

2 Deep Reinforcement Learning

2.1 Reinforcement Learning

Reinforcement learning (RL) is framework for solving and modeling sequential decision-making problems in dynamic and uncertain environments. RL models learn to solve a problem by trying different actions in the specified environment and then discovering the reward that different actions yield. The learner, called an agent, thus must be able to observe the state, take actions, and have a goal in maximizing the reward. In the end, the agent should learn the optimal policy function that maps states to actions to maximize cumulative reward [11]. Deep reinforcement learning (DRL) is an extension of RL where deep learning is integrated into RL framework. By using deep neural networks for approximating functions DRL models can operate in even more complex systems.

There are two categories of RL, model-based and model-free. The difference between these two is model-based RL learns the optimal policy from the dynamics of the environment and model-free RL learns the policy by trial and error. Model-free RL is well suited for complex, non-stationary and often partially observable problems, such as financial applications. Compared to model-based RL, model-free RL has significant advantage in finance, where the dynamics of the environment are largely unknown and may even change over time due to regime shifts or macroeconomic factors.

A growing number of studies and articles have considered the use of RL models in finance. One of the more common uses for RL methods in finance is portfolio allocation. One example using model-based DRL for portfolio optimization is by Yu et al. [12], where asset prices were predicted using historical and synthetic data generated using Generative Adversarial Network. A portfolio trading strategy based DQN was proposed by Park et al. [13], where the model either holds, buys or sells a prespecified quantity of the asset at each point in time. In the study, the DQN model outperformed the benchmark strategies. Yang et al. [14] combined multiple DRL models into an automated stock trading strategy. Du et al. [9] applied DRL into hedging options position and shows that the model learns how to hedge the option position in a realistic setting, where PPO performed the best out of the tested models. Cao et al. [15] used Q-learning and DDPG for hedging a short call option and the results showed that the DRL approach had considerably less transaction costs than delta hedging. Similarly, in [16] Buehler et al. use RL to successfully create optimal hedging strategy in the Heston model that was tested with and without market frictions. Their results also showed that the learned hedging strategy outperformed delta hedging with market frictions.

2.2 Markov Decision Process

Most reinforcement learning problems are framed as discrete time Markov Decision Processes (MDPs), which model decision-making in uncertain and dynamic environments. An MDP consists of the decision maker and learner called the agent that interacts with an environment. The environment is characterized by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward space $\mathcal{R} \subset \mathbb{R}$, and a transition probability function p. At each time step $t = 0, 1, 2, ...$, the agent observes the current $S_t \in \mathcal{S}$, chooses an action $A_t \in \mathcal{A}$, receives a reward $R_{t+1} \in \mathcal{R}$, and transitions to a new state $S_{t+1} \in \mathcal{S}$. This sequence is either continuing or ends at a terminal time step T, in which case the sequence is called an episode. The dynamics of the environment are defined by the transition probability function p that denotes the probability of transitioning to state s' and receiving reward r, given that the agent took action a in state s at the previous time step, i.e.,

$$p(s', r \mid s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a].$$

The transition probability function means that the current state and the action contain all the necessary information for predicting the next state and expected reward for that action. If an environment has these dynamics it is said to have Markov property[11]. Markov property is a key principle in reinforcement learning as it simplifies the modeling of sequential decision making problems as the algorithms can estimate the values of states and actions without the need for tracking the entire history of the process.

In RL an agent interacts with an environment in discrete time steps with the goal of maximizing a cumulative return, which represents the total accumulated reward. Formally, the return at time step t, denoted by G_t, is defined as the sum of future rewards

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where R_{t+k+1} is the reward received k steps into the future and $\gamma \in [0, 1]$ is the discount factor that determines the present value of future rewards. The discount factor describes how much the agent prefers immediate rewards compared to future rewards.

The agent's behavior is described by policy π, which specifies the probability distribution over actions given states. Formally it is defined as a mapping

$$\pi(a \mid s) = P[A_t = a \mid S_t = s],$$

where $\pi(a \mid s)$ gives the probability of taking action a in state s. The quality of a policy is measured by the value function it induces. A policy is optimal if it yields highest possible expected return from all states [11]. Thus, the state-value function $v_\pi(s)$ for policy π [11], when starting in s and following π, is the expected return. It can be defined as

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s].$$

Similarly action-value function $q_\pi(s, a)$ for policy π [11], gives the expected return starting from s, taking action a and following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a].$$

To formalize the maximum of the cumulative reward over time, optimal value functions are defined. The optimal value functions represent the best possible performance achievable from any given state or state-action pair, assuming optimal behavior by the agent. The optimal state-value $v_*(s)$ and action-value $q_*(s, a)$ functions can then be defined in terms of v_π and q_π as

$$v_*(s) = \max_\pi v_\pi(s)$$
$$q_*(s, a) = \max_\pi q_\pi(s, a),$$

for all $s \in S$ and in the case of action-value function also for all $a \in A$. Thus the optimal policy π_* is defined as either the policy that maximizes the state-value function for all states, or the action-value function for all states and actions. The optimal value functions are essential to solving RL problems because they encapsulate the best possible performance from any given state or state-action pair. Optimal policy can be directly derived by maximizing expected return once either of the optimal value functions is known. In practice, these functions are often approximated using iterative updates or function approximation techniques.

The relationship between the value of a state and the values of its successor states are expressed as The Bellman equations. For a given policy π [11], the Bellman expectation equation state-value function v_π

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right],$$

for all $s \in S$. This equation states that the expected return from state s under policy π is equal to the expected immediate reward plus the discounted reward from the next state onward. Similarly, the Bellman expectation equation for the action-value function $q_\pi(s, a)$ [11] is

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right],$$

for all $s \in S$ and $a \in A$. The Bellman equations provide a mechanism for recursively calculating the expected return for a given policy. For the optimal value functions, the Bellman equations evolve into Bellman optimality equations, that represent the maximum expected return from any state or state-action pair, as opposed to a specific policy. The Bellman optimality equation for state-value function [11]

$$v_*(s) = \max_a \mathbb{E} \left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\right].$$

The Bellman optimality equation for action-value function [11]

$$q_*(s, a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a\right]. \tag{1}$$

The optimal value function $v_*(s)$ is not dependent on any policy π. Since the Bellman optimality function holds for all $s \in S$, it forms a system of nonlinear equations. Solving this system results in the optimal value function v_*, from which the optimal policy can be derived by selecting actions that achieve the maximum in the Bellman optimality equation.

In RL Bellman equations are usually used in value iteration and policy iteration algorithms. In policy iteration, the Bellman expectation equation is used to evaluate a given policy and then the policy is improved by acting greedily with respect to the estimated value functions. In value iteration, the Bellman optimality equation is used iteratively in improving the estimates of $v_*(s)$ until they converge to the optimal value function. Model-based reinforcement learning algorithms, such as policy iteration and value iteration, require full knowledge of the environment, specifically the transition probabilities and reward function. When the environment dynamics are unknown, model-based reinforcement learning algorithms become infeasible. In such cases model-free reinforcement learning algorithms that learn approximations of the value and policy functions from experience are required.

2.3 Deep Learning

When the state space S becomes large or continuous, solving for the exact optimal policy or action-value function is generally intractable. To address this challenge, the most common approach is to use function approximation, where the target function is estimated from data. In deep learning, artificial neural networks (ANNs) are used for approximating the functions. ANNs are computational models inspired by the structure of biological neural networks, where artificial neurons are connected through weighted links analogous to synapses. These networks can be composed of multiple layers, enabling them to learn hierarchical representations of data. When there are several hidden layers, the approach is referred to as deep learning. A key property of neural networks is that they are universal function approximators, which means that given a sufficiently large architecture they approximate any continuous function to arbitrary precision [17]. This feature makes them especially suitable for reinforcement learning tasks where the true value functions are unknown.

To illustrate the structure and information flow in a neural networks, Figure 1 shows an example of a simple feedforward neural network (FNN). The diagram consists of an input layer, two hidden layers and an output layer, with each circle representing neurons. Arrows denote the weighted connections between neurons. When every neuron in one layer is connected to every neuron in the next, all layers are fully connected. Each neuron computes the weighted sum of the outputs from the previous layer. Often a constant term, called the bias, is added to the sum. This sum is then passed through an activation function that yields the output of the neuron. The lack of loops in the graph characterizes that the network is a feedforward network, in which the information flows from input layer to output layer without feedback loops.

The fundamental unit of feedforward neural network is a fully connected layer, which performs an affine transformation followed by the application of an activation function. Mathematically, the output of a fully connected layer can be described as matrix multiplication:

$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where, **y** is the output vector, g is the activation function applied element-wise, **W** is the weight matrix, **x** is the input vector and **b** is the bias vector.

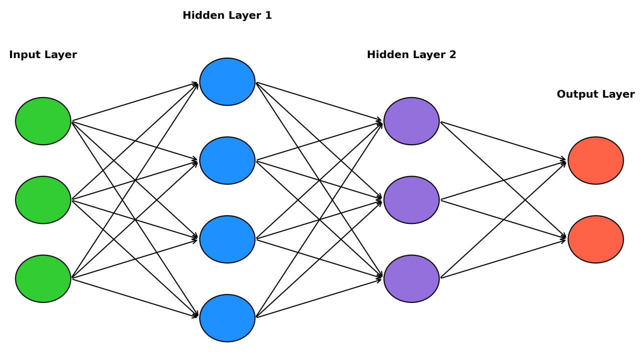

Figure 1: Illustration of a fully connected feedforward artificial neural network. This network consists of three input nodes connected to a hidden layer with four nodes that is connected to another hidden layer with three nodes, which is connected to output layer with two nodes.

Fully connected layers are just one example of deep learning architectures. There are other types of layers such as convolutional layers and recurrent layers. If a neural network contains convolutional layer, it is called convolutional neural network (CNN). In a similar way if a neural network contains recurrent layer it is called a recurrent neural network (RNN). CNNs are best-suited for tasks involving grid-like data, such as image recognition or audio processing [18]. In a CNN, layers consist of convolutional filters that learn local spatial hierarchies in the data. For example when audio waveform is considered as the input, only nearby time frames are connected to form a node in the following layer. This is effective since audio signal often exhibit local temporal dependencies. Convolutional layers can apply filters that detect local features, enabling the network to detect repeating patterns such as syllables or notes, regardless of their position in the input.

RNNs, on the other hand, are designed for sequential data, such as time series or natural language. RNNs have connections that form loops, allowing information to be passed from one step to the next. These loops enable RNNs to have memory, which makes them capable of modeling time-dependent relationships. For example, when processing time series data such as stock prices, the RNN captures temporal dependencies into hidden states that act as the memory. Each neuron in an RNN receives information not only from the current input but also from its own output at the previous time step. This feedback loop allows the network to remember relevant past data and use it to inform current predictions. In stock price context, this could mean momentum or volatility clustering.

Across all types of deep learning networks, activation functions are essential in introducing nonlinearity to the model [18]. Without activation functions the network would simply perform linear transformations, which would significantly limit its ability to capture complex patterns in the data. Common activation functions include sigmoid function $\sigma(x)$, hyperbolic tangent function $\tanh(x)$ and the rectified linear unit (ReLU):

$$\begin{aligned}\sigma(x) &= \frac{1}{1+e^{-x}}\\ \tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}}\\ \mathrm{ReLU}(x) &= \max(0, x).\end{aligned}$$

Training deep networks requires adjusting the weights and biases using an objective function, often called a loss function. Common loss functions are mean squared error for regression and cross-entropy for classification. The optimization process is typically performed using some version of Stochastic Gradient Descent (SGD). The goal is to minimize a loss function $J(\theta)$ by optimizing the parameter vector θ that includes all the weights and biases of the neural network. SGD is an iterative optimization method, that samples a small batch of training data at each iteration k and the gradient of the loss function $\nabla_\theta J(\theta_k)$ with respect to the parameters is computed. The parameters are then updated in the direction that reduces the loss most rapidly, scaled by the learning rate α that controls the size of the update steps. The update rule can be written as [18]

$$\begin{aligned}\theta_{k+1} &= \theta_k - \alpha \nabla_\theta J(\theta_k)\\ \text{where}\quad J(\theta_k) &= \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(f(x_i;\theta_k), y_i).\end{aligned}$$

By repeating this update process many times, the network gradually learns the optimal values for its parameters.

2.4 Reinforcement Learning Methods

In classical reinforcement learning, algorithms often rely on tabular representations that save the value function estimates for each state explicitly. However, in large or continuous state spaces tabular methods become infeasible due to the curse of dimensionality [19]. To address this limitation, deep reinforcement learning integrates deep neural networks as function approximators, allowing agents to generalize across similar states and learn directly from high dimensional inputs such as financial time series.

DRL methods are commonly categorized into model-based and model-free approaches. Model-based methods either try to learn or are given an explicit model of the environment's transition dynamics and reward function. Model-based algorithms can predict the consequences of actions, allowing the agent to choose the optimal action by simulating future states and rewards. As a result model-based approaches learns with higher sample efficiency and with less training time compared to model-free methods. However, accurately modeling the environment in a realistic way is often difficult and environments with changing or non-stationary dynamics to models that perform well during training but poorly generalize out of sample.

In contrast, model-free methods learn optimal policies or value functions directly from experience without explicitly constructing a model of the environment's dynamics. The typical way to estimate how good actions are in given states is by interacting with the environment using trial-and-error and updating their knowledge based on observed rewards and transitions. Model-free methods are generally easier to implement and can be effective when modeling complex environments, but they usually require large amounts of data. The two most common examples of model-free methods are Q-learning and policy optimization.

DRL methods can still be divided into value-based methods and policy-based methods. Value-based methods focus on learning a value function to estimate the expected return of being in a certain state or taking a certain action. They aim to find an optimal policy indirectly by first learning the value functions. Commonly the action-value function $Q(s, a)$ is used to estimate the expected cumulative reward when taking action a in state s and following the policy thereafter. Following this process an approximation of the optimal Q-function $Q^*(s, a)$ is learned and the optimal policy can be derived by selecting the actions that maximize the Q-value

$$\pi^*(s) = \arg\max_a Q^*(s, a).$$

A common example of a value based method is the Q-learning algorithm, which updates the Q-function iteratively using the Bellman equation (1) as a target [11]. The update rule in Q-learning is based on temporal difference (TD) learning, where the Q-value is adjusted using the TD error

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right],$$

where $\alpha \in (0, 1]$ is the learning rate and $\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$ is the TD error, which measures the discrepancy between the estimated Q-value and the new target based on observed reward and future predictions. TD error is the learning signal used to update the Q-function [11]. The learning rate controls the amount of new information used per iteration. Smaller α leads to slower but more stable learning, while large α learns faster but has larger instability risk. Q-learning is an off-policy method, which means it directly approximates the optimal action-value function Q^* independently of the agent's current policy [11].

In contrast, policy-based methods directly represent the policy through a parameterized function $\pi_\theta(a \mid s)$, where θ defines the policy's behavior. Then the parameters θ, such as weights of a neural network are optimized to maximize the expected cumulative reward. The most common policy-based method is the policy gradient method, which will be presented in detail in the following subsection. Strengths of the policy-based methods include the ability to handle stochastic policies, which can be useful in environments with partial observability or multiple optimal actions, where the typically deterministic value-based methods struggle. Compared to value-based methods, policy-based methods are generally more stable during training, but they tend to be less sample-efficient and may converge more slowly.

Given the complementary strengths and weaknesses of value-based and policy-based methods, a hybrid approach between the two called actor-critic method has emerged. Actor-critic architecture consists of an actor, that represents the policy by selecting actions, and a critic, which estimates the value function to evaluate the actions taken by the actor. The critic's evaluation provides a learning signal that reduces the variance of policy gradient estimates, leading to more stable and efficient training. Actor-critic methods are the basis of many advanced deep reinforcement learning algorithms.

2.5 Policy Gradient Methods

Policy gradient (PG) methods are a class of reinforcement learning algorithms that use gradient ascent to learn the optimal policy. Instead of learning a value function to derive a policy PG methods model the policy explicitly as a probability distribution over actions given states. As mentioned in previous section a policy is typically parameterized as $\pi_\theta(a \mid s)$, where θ represents the parameters of the model, often the weights of a neural network. The objective of the agent is to find parameters θ that maximize the expected return, defined as the cumulative reward over time. Formally, objective function is

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right],$$

where R_t is the reward at time t, and $\gamma \in [0, 1]$ is the discount factor that determines how much immediate rewards are preferred over distant future rewards [11].

To optimize the objective function $J(\theta)$, policy gradient methods employ the policy gradient theorem [20], which provides a tractable expression for the gradient of the expected return with respect to the policy parameters. The theorem states that

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s)\, Q^\pi(s, a)\right], \tag{2}$$

where $Q^\pi(s, a)$ is the action value function under policy π, representing the expected return after taking action a in state s and following policy π thereafter. This formulation allows for estimating the policy gradient using sampled trajectories from interaction with the environment, without needing to differentiate through the environment's dynamics. The term $\nabla_\theta \log \pi_\theta(a \mid s)$ is called the score function that highlights how small changes in the policy parameters affect the log-probability of selecting an action. Since it is multiplied with corresponding Q-values, higher Q-value assigns higher weight to more rewarding actions. This ensures that the policy is updated in a direction that increases the likelihood of actions yielding higher long-term returns.

In practice, the policy parameters are updated through stochastic gradient ascent according to the gradient estimate

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta).$$

One of the earliest policy gradient algorithms is the REINFORCE algorithm [21]. It is a Monte Carlo method that uses sampled returns to approximate the policy gradient. It replaces the action-value function $Q^\pi(s, a)$ in the policy gradient theorem 2 with the empirical return $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$ and by using stochastic gradient ascent, the update rule becomes

$$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t.$$

In practice the parameters are thus updated using

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t.$$

One downside of policy gradient estimates is that they can have high variance, which can slow or destabilize learning. To address this, variance reduction techniques are often used.

One common approach is to subtract a baseline $b(s)$ from the Q-value, leading to modified gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a \mid s) \left(Q^\pi(s, a) - b(s) \right) \right].$$

A widely used choice for the baseline is the state-value function $V^\pi(s)$, which estimates the expected return from state s. This advantage-based formulation reduces variance of the gradient estimate without introducing bias, making policy updates more stable and efficient [11].

2.6 Proximal Policy Optimization

Policy gradient methods such as REINFORCE and even more advanced actor-critic variants can suffer from high variance and unstable updates, especially when using deep neural networks to represent the policy. Proximal Policy Optimization (PPO), introduced by [22], addresses these issues by balancing the ease of implementation, sample efficiency, and reliable policy improvement. The central motivation behind PPO is to avoid large, destabilizing policy updates that can degrade performance. Traditional policy gradient methods may take large steps during optimization, potentially collapsing the policy. PPO uses a clipped objective function to limit the difference between the new policy and the old policy to ensure more stable and reliable learning.

One of the first methods addressing the instability in policy gradient updates by controlling the size of policy updates was Trust Region Policy Optimization (TRPO), introduced in [23]. It formulates a constrained optimization problem that maximizes a surrogate objective while ensuring that the Kullback-Leibler (KL) divergence [24] between the new and old policies remains within a trust region.

$$L^{\text{TRPO}}(\theta) = \mathbb{E}_{s \sim \rho_{\theta_{\text{old}}}, a \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} Q_{\theta_{\text{old}}}(s, a) \right]$$

$$\text{s.t.} \quad \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot|s) \parallel \pi_{\theta}(\cdot|s)\right)\right] \leq \delta,$$

where $\pi_\theta(a|s)$ is the new policy, $\pi_{\theta_{\text{old}}}(a|s)$ is the old policy, $Q_{\theta_{\text{old}}}(s,a)$ is the estimated action-value function under the old policy, $\rho_{\theta_{\text{old}}}$ is the state distribution under old policy, D_{KL} is the KL divergence and δ is a parameter that limits the KL divergence.

While effective, TRPO is complex to implement and computationally expensive. Thus, in response PPO was developed. PPO simplifies TRPO by using a clipped surrogate objective that avoids the need for complex constraints or second order methods. The PPO objective function is

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right],$$

where $r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, $\hat{A}_t$ is the advantage function estimate $\hat{A}_t = Q_\pi(s_t, a_t) - V_\pi(s_t)$, and ϵ is a small positive hyperparameter. The clipping term clip $(r_t(\theta), 1-\epsilon, 1+\epsilon)$ limits the maximum improvement of the new policy to avoid large policy changes that might hurt performance.

In policy gradient methods, accurately estimating the advantage function is crucial for effective learning. The advantage function, A_t, quantifies how much better an action a_t taken in state s_t is compared to the expected value of the state. In other words, it measures the relative benefit of an action over the baseline policy, guiding the policy update toward actions that improve expected returns. Estimating advantages directly from sampled trajectories can be noisy and high variance, which can destabilize learning. To mitigate this, Generalized Advantage Estimation (GAE) [25] is often used. GAE balances the tradeoff between bias and variance in advantage estimates by computing the advantage as a weighted sum of TD residuals over multiple time steps

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\,\delta_{t+l}^V,$$

where the TD residual δ_t^v is defined as

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t),$$

in which r_t is the reward at time t, $V(s_t)$ is the estimated value of state s_t, $\gamma \in [0,1]$ is the discount factor controlling the importance of future rewards and $\lambda \in [0,1]$ is a parameter controlling the bias-variance tradeoff. When $\lambda = 0$, GAE reduces to the one-step TD error that yields low variance but potentially biased estimates. Conversely, when $\lambda = 1$, it corresponds to Monte Carlo estimation, which is unbiased but has high variance. By tuning λ stable and efficient policy updates can be achieved.

PPO training follows an iterative process of alternating between data collection and policy improvement. Each cycle begins by executing the current policy within the environment to generate a set of experience trajectories. These trajectories form the foundation for policy updates and are used to estimate how much better or worse the chosen actions were compared to the expected value of each state. Once the trajectories are collected, the algorithm computes advantage estimates, often using GAE, and evaluates the probability ratio between the new and old policies. These quantities are then used to form the clipped surrogate objective, which is maximized using stochastic gradient ascent across several mini-batches and epochs. In parallel, the value function is refined by minimizing the mean squared error between the predicted values and the empirical returns, ensuring consistent updates for both the actor and the critic.

A central aspect of PPO training is the reuse of a fixed set of trajectories collected under current policy. Instead of collecting new data every small update, PPO performs multiple optimization steps on the same batch, improving sample efficiency. To avoid divergence from the original data distribution and to promote stability, the objective includes the clipped probability ratio, which prevents the new policy from moving too far away from the old one.

3 Liquidity Provision

3.1 Background

In financial markets liquidity refers to the ease with which an investment can be bought or sold without impacting its market price. Characteristics of a liquid market include tight bid-ask spreads, deep order books and minimal slippage, the difference between the expected price of a trade and the price at which the trade is executes, for orders of any size. High liquidity enables quick and efficient transactions, which leads to accurate pricing and more stable markets. In addition, in liquid markets transaction costs are smaller and capital can move across markets easily.

In traditional finance liquidity is often provided using limit order books, where market makers place limit orders on both sides of the market, profiting from the spread. Market makers must actively manage their inventories and have sufficient strategies for surviving different market conditions, such as news catalysts and volatility changes. In contrast, DeFi introduces automated market making, where liquidity provision is implemented using smart contracts called liquidity pools. Smart contracts are self-executing programs deployed on the blockchain that automatically enforce the trading rules and hold assets without intermediaries. Liquidity pools are smart contracts that contain reserves of two tokens, digital assets such as ETH or Bitcoin (BTC), that can be transferred and traded on the blockchain. This represents a fundamentally different mechanism from traditional finance, as liquidity is supplied directly by users who deposit cryptocurrency tokens into these smart contracts to enable trading.

In decentralized exchanges (DEXs) like Uniswap, LPs make trading assets without the need for centralized intermediaries or order books possible. To clarify the ecosystem in which liquidity provision takes place, Figure 2 illustrates the main actors in a decentralized exchange such as Uniswap, their objectives, and the transactions between them.

The Uniswap protocol functions as a decentralized exchange that hosts many liquidity pools. Liquidity providers deposit token pairs, known as trading pairs, into these pools to enable trading and earn a share of transaction fees. A trading pair defines the two tokens that can be exchanged against each other, for example ETH and BTC. Traders and arbitrageurs interact with these pools by executing swaps that change the pool's reserves—the quantities of each token held in the pool—and, in turn, its price. Typically, an LP deposits equal values of both tokens in the trading pair into a liquidity pool governed by a predefined pricing function. These pooled assets facilitate trading automatically through the underlying smart contract. The reward for supplying market liquidity is a proportional share of the transaction fees generated by the pool. Essentially this mechanism allows anyone to become an LP and contribute to the market infrastructure. However, it is worth noting that, on average, LPs tend to underperform simply holding the tokens they provide [4, 5].

Despite concentrated liquidity only being introduced in Uniswap v3 in 2021, numerous studies have examined liquidity provision. Cartea et al. [5] study the optimal liquidity provision strategy from the perspective of the width of the liquidity position by maximizing LPs wealth by balancing the losses from providing liquidity and the

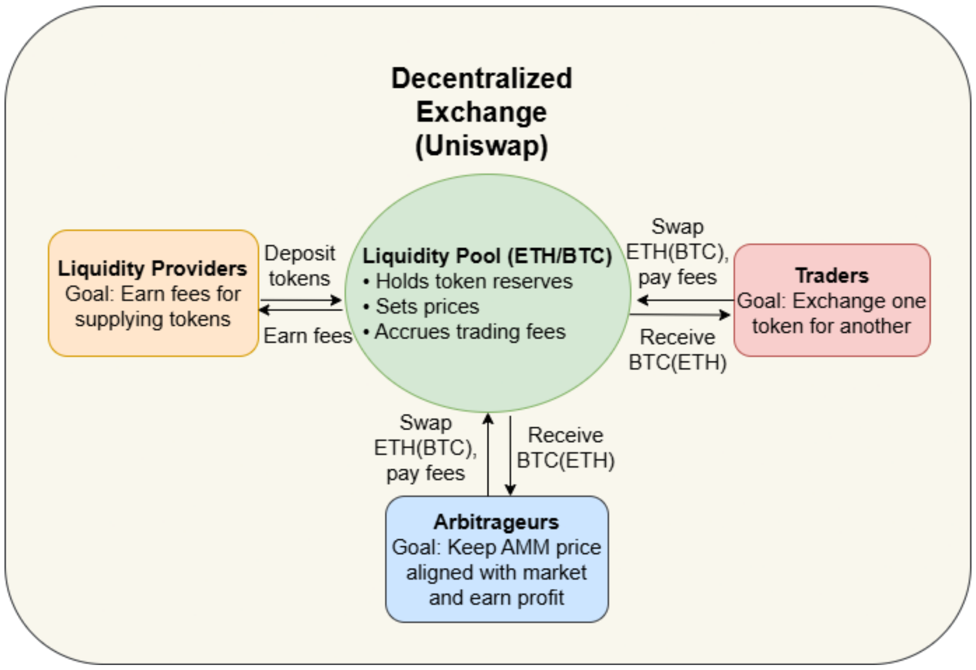

Figure 2: Actors and interactions within a decentralized exchange (Uniswap). The Uniswap protocol provides a decentralized environment on the Ethereum blockchain where all market activity occurs through smart contracts. Liquidity providers deposit token pairs into a shared liquidity pool, enabling traders and arbitrageurs to exchange tokens. Traders swap one asset for another and pay transaction fees, while arbitrageurs maintain price alignment with external markets. Accrued fees are distributed back to liquidity providers as a compensation for supplying liquidity.

fees earned. The risk and return trade-off of Uniswap v3 liquidity providers are studied Heimbach et al. [6], who suggest that retail traders should not chase high yields as there are substantial risks involved. A dynamic liquidity provision model using neural networks is provided in [7]. Fan et al. [26] study the optimal liquidity provision and considers the trade-offs from position width and how it affects the fee revenue and impermanent loss. Successful hedging of liquidity positions using options are studied in [27, 28, 29]. These studies show that the hedging or replication of the liquidity positions is possible when there is liquid options market available. Angeris et al. [30] demonstrate that liquidity provision positions in concentrated AMMs can be constructed to replicate a variety of payoff profiles, including those resembling traditional financial instruments such as covered calls.

This thesis focuses on how to hedge liquidity positions by shorting the replica of the liquidity position using futures and deep reinforcement learning. This thesis only attempts to optimize the hedge by minimizing the cost and does not consider the fee revenue or optimal allocation of LPs. A similar study by Zhang et al. [8] implements a DRL algorithm for LP decision-making. It considers loss-versus-rebalancing (LVR), a metric proposed by [31], and optimizes the LP positions by updating the width of the liquidity pools in a way that minimizes LVR. LVR is defined as the difference in value of the original liquidity position and replicating portfolio and can be referred to as the costs for hedging the liquidity position. Cartea et al. present [32] another measure called predictable loss (PL) that is similar to LVR but adds a component of investing excess capital into risk-free account. Fukasawa et al. [33] demonstrate that by using a model-free hedging strategy and using no-arbitrage condition LVR can be hedged completely.

3.2 Automated Market Makers

Automated market makers (AMMs) are algorithms that automate the process of executing trades on DEXs without relying on any centralized intermediaries. Using AMMs, users can trade directly with a liquidity pool that manages and prices the tokens in them according to a pricing function. AMMs allow continuous liquidity and open participation, while being simple enough for arbitrageurs to synchronize the DEX price with a price from a centralized exchange [34, 35]. The most well-known AMM model is the constant product formula

$$x \cdot y = k, \tag{3}$$

where x and y represent the reserves of two tokens in the pool, and k is a constant. As the product of x and y remains always constant, it imposes a non-linear price curve. When a trader buys one asset from the pool it decreases the relative reserve of that asset, leading to an increase in its price due to the inverse relationship with the remaining quantity. Constant product formula defines the spot price at any point on the price curve as

$$P = \frac{y}{x}. \tag{4}$$

Because the price is calculated using predefined formula, it ensures that the liquidity pool always offers a price for the tokens in all market conditions. However, due to the convex shape of the pricing curve, larger trades lead to disproportionately higher slippage, which means that traders often get a slightly worse price than expected. This creates an implicit spread between buy and sell prices, even though no order book exists. This is a fundamental trade-off in CFMM design, balancing continuous liquidity with increasing costs for large volume trades [35].

In constant product automated market makers (AMMs), liquidity providers must deposit equal dollar values of each token into the pool. This ensures that the pool starts and remains balanced in terms of value exposure. More specifically, if the external market prices of tokens x and y are p_x and p_y, then the deposited quantities x and y must satisfy the equality

$$x \cdot p_x = y \cdot p_y.$$

This equal dollar value allocation is preserved by the constant product mechanism $x \cdot y = k$, which continuously adjusts the token ratios as prices change. As a result, the total value of each token reserve remains equal, and LPs always hold a portfolio that is equally weighted in dollar terms.

To illustrate how the reserves evolve with price under this model, consider that the AMM must always satisfy the condition in equation (3), even as the price P changes. Using the spot price relation (4), we can express the reserves x and y as functions of the price. Solving for x and y yields

$$\begin{aligned} x(P) &= \sqrt{\frac{k}{P}}, \\ y(P) &= \sqrt{kP}. \end{aligned} \tag{5}$$

These expressions show that as the price of token x increases, the pool holds less of token x and more of token y, preserving the constant product invariant. Liquidity providers who initially deposited equal-value amounts of both tokens will see their position shift over time as the price fluctuates, gradually accumulating more of the depreciating asset and less of the appreciating one. This rebalancing occurs automatically, resulting in a continuous dynamic dollar-cost averaging mechanism inherent to the AMM structure.

Arbitrageurs are responsible for keeping AMM prices aligned with global market prices. Angeris et al. [34, 35] show that the prices of Uniswap are close to prices in other exhanges. When arbitrageurs spot discrepancies between AMM price and external price, they push the AMMs price towards the correct price, while profiting from the difference. This mechanism makes sure that AMMs are market-responsive but can stay passively managed [35]. However, in traditional AMMs there are significant trade-offs. Uniformly distributing liquidity across the entire price curve leads to inefficient capital allocations, especially at the far ends of the price curve. Since the market price is unlikely to move uniformly across the entire price curve, a significant portion of the capital is either rarely or never utilized. As the capital is not used efficiently, it leads to higher than necessary slippage and lower fee revenue per deposited capital for LPs. These properties led to the development of more advanced AMMs, such as Uniswap v3, that improves capital efficiency by allowing LPs to concentrate their liquidity to specified price intervals and making more advanced liquidity provision strategies possible for LPs [3].

3.3 Concentrated Liquidity

Uniswap v3 introduced concentrated liquidity that extends the AMM model by allowing LPs to allocate token reserves x and y (referred to as real reserves) to a bounded price interval $[P_a, P_b]$, where P_a is the lower bound of the interval and P_b is the upper bound of the interval. This means that liquidity is provided only for trades that occur within a chosen price range, instead of being spread evenly across all possible prices.

Uniswap v3 discretizes prices into a set of ticks, where the price corresponding to the ith tick is given by 1.0001^i. The tick spacing, which determines the smallest permissible increment between ticks and thus the minimum width of a liquidity position's active price interval, depends on the pool's fee tier [3]. For example, pools with a 1% fee tier have a tick spacing of 200 ticks, pools with a 0.3% fee tier have a tick spacing of 60 ticks, and pools with a 0.05% fee tier have a tick spacing of 10 ticks.

This design ensures that the minimum price range of a liquidity position approximately corresponds to twice the fee tier, thereby aligning the granularity of liquidity provision with the associated trading costs.

When an LP provides liquidity to a liquidity pool, it is referred to as a position. Unlike prior AMM designs that distribute liquidity uniformly across all prices, concentrated liquidity allows LPs to confine their capital to a specific price range, similar in spirit to placing limit orders around expected price levels. Within this range, the position behaves like a leveraged constant-product pool. The reserves within this interval are referred to as virtual reserves, representing the notional token amounts that would yield the same liquidity and pricing behavior in a traditional constant-product pool [3].

The following equations formalize the relationship between liquidity, price range, and token amounts. They describe how the pool's reserves evolve as the market price moves within or beyond the active range.

In Uniswap v3, the liquidity L is a parameter that determines how much token inventory the AMM provides at a given price range $[P_a, P_b]$. In Uniswap v3, liquidity is defined as

$$L = \sqrt{k},$$

where k is the constant-product invariant that would hold if the position were a full-range (v2-style) AMM. The square-root definition ensures that liquidity scales linearly with deposited capital and that multiple LP positions can be added together simply by summing their liquidity values. Liquidity is associated with the concept of virtual reserves x_v and y_v, defined as

$$\begin{aligned} x_v &= x + \frac{L}{\sqrt{P_b}} \\ y_v &= y + L\sqrt{P_a}. \end{aligned} \tag{6}$$

Virtual reserves do not correspond directly to the actual token amounts held by the LP. Instead, they allow Uniswap v3 to generalize the constant-product model to a setting where liquidity is provided only within a finite price interval. The liquidity parameter L governs the "depth" of the position: higher L corresponds to more capital concentrated in the range and a smaller price impact per trade. The product of virtual reserves satisfies

$$x_v \cdot y_v = L^2, \tag{7}$$

which ensures deterministic pricing behavior within the active range.

The price P, defined as the ratio of virtual reserves, is given by

$$P = \frac{y_v}{x_v} = \frac{y + L\sqrt{P_a}}{x + \frac{L}{\sqrt{P_b}}} \tag{8}$$

Combining the constant product condition shown by equation (7) of virtual reserves and the price equation (8) we can express the virtual reserves as a function of the current price P and liquidity L

$$x_v = \frac{L}{\sqrt{P}},$$
$$y_v = L\sqrt{P}.$$

By substituting the virtual reserve definitions into equations (6) and solving for the real reserves x and y we obtain the following expressions as functions of liquidity L, current price P, and the bounds of the active range $[P_a, P_b]$

$$x = L\left(\frac{1}{\sqrt{P}} - \frac{1}{\sqrt{P_b}}\right),$$
$$y = L\left(\sqrt{P} - \sqrt{P_a}\right).$$

When the price moves outside of the bounds of the active range $[P_a, P_b]$, the position consists of only one asset. If $P \leq P_a$ the position consists of only tokens x and if $P \geq P_b$ the position consists of only tokens y. Thus the token amounts can be formulated piecewise by

$$(x, y) = \begin{cases} \left(L\left(\frac{1}{\sqrt{P_a}} - \frac{1}{\sqrt{P_b}}\right),\ 0\right), & \text{if } P \leq P_a \\ \left(L\left(\frac{1}{\sqrt{P}} - \frac{1}{\sqrt{P_b}}\right),\ L\left(\sqrt{P} - \sqrt{P_a}\right)\right), & \text{if } P_a < P < P_b \\ \left(0,\ L\left(\sqrt{P_b} - \sqrt{P_a}\right)\right), & \text{if } P \geq P_b. \end{cases} \tag{9}$$

This piecewise formulation illustrates the changing token exposure of a position as the price moves across and beyond the active range. Within the range, both tokens are present; outside it, the position holds only one token depending on the direction of price movement.

3.4 Risk and Exposure of Liquidity Providers

Liquidity provision in DEXs offers fee income opportunities, but it also exposes LPs to unique and complex risks. Unlike passive asset holding, LPs commit capital into an AMM mechanism, which subjects their positions to market dynamics that often lead to underperformance compared to simply holding the underlying assets [4, 5]. This underperformance is called impermanent loss and occurs because the AMM automatically rebalances the token holdings, selling assets as they rise in price and buying them as they fall, opposite to what a typical investor would prefer. This section outlines the key risk components inherent to concentrated liquidity provision, which serve the basis for the hedging strategies developed later in this thesis.

To understand the risk profile of a liquidity provider (LP), it is essential to analyze how the fiat-denominated value of a liquidity position evolves as the market prices of the two underlying assets change. Since the token prices are usually denominated in US dollars, we will also denominate the values of the liquidity positions in dollars. The composition and valuation of the position are directly determined by the underlying AMM mechanism and the specified price range of the position. We begin by considering the classical full-range liquidity case such as in Uniswap v2, and then extend the analysis to the more general concentrated liquidity model introduced by Uniswap v3.

The total dollar value of a full-range liquidity position, a position that provides liquidity to all prices along the price curve, can be expressed as the sum of values of the individual token reserves x and y, each weighted by its respective market price p_x and p_y respectively. Therefore the total dollar value of a full range liquidity position is

$$V_{\text{v2,usd}}(P, p_x, p_y) = x(P) \cdot p_x + y(P) \cdot p_y. \tag{10}$$

By substituting the token reserve equations from equation (5) into this expression and assuming a constant product k, we obtain

$$V_{\text{v2,usd}}(P, p_x, p_y) = \sqrt{\frac{k}{P}} \cdot p_x + \sqrt{kP} \cdot p_y, \tag{11}$$

This function gives the total dollar value of the position as a function of the price ratio $P = p_x/p_y$ and the individual token prices. It illustrates the non-linear relationship between the position's dollar value and changes in the prices of the tokens. Figure 3 illustrates how the full-range liquidity position dollar value behaves when the price of token x changes. Figure 3 also demonstrates that large increases in the price of one token result in dampened increases in the dollar value of the liquidity position. We can see that the position doubles in value when p_x quadruples, when the position increases in value by close to 20% when p_x increases by 20%. Conversely, large decreases in the price of one token lead to increasing losses, which grow larger as the token price declines further, which is seen by the steepening slope when the price is moving closer to zero.

In concentrated liquidity models, liquidity providers concentrate their capital within a finite price interval $[P_a, P_b]$, rather than distributing liquidity uniformly across the entire price axis in full-range liquidity models such as Uniswap v2. Although the constant product formula still governs the pool dynamics locally within this interval, liquidity is now allocated more efficiently, focusing on the specified price range.

As established earlier in the piecewise function (9), the token amounts depend on the position of the current price relative to the specified range. When the price is outside the active range, the position consists entirely of one token: only token x when $P \leq P_a$, and only token y when $P \geq P_b$. Within the active range $P_a < P < P_b$, the position contains both tokens. The following piecewise function describes how the total dollar value of a concentrated liquidity position changes depending on whether the market price is below, inside, or above the active price range. Using the expressions for token holdings in each region and multiplying them by the token prices p_x and p_y, the value function can be written in piecewise form as

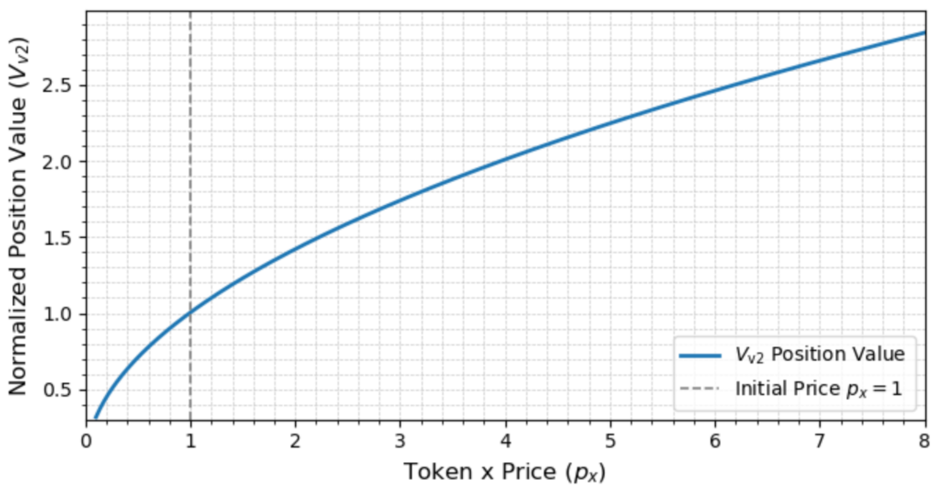

Figure 3: Normalized dollar value of a full-range liquidity position with different varying prices of token x. Price of token y p_y was kept constant to illustrate the non-linear relationship between the liquidity position dollar value and the price of a single token.

$$V_{\text{v3,usd}}(L, P, P_a, P_b, p_x, p_y) = \begin{cases} L\left(\frac{1}{\sqrt{P_a}} - \frac{1}{\sqrt{P_b}}\right) p_x, & \text{if } P \le P_a \\ L\left[\left(\frac{1}{\sqrt{P}} - \frac{1}{\sqrt{P_b}}\right) p_x + \left(\sqrt{P} - \sqrt{P_a}\right) p_y\right], & \text{if } P_a < P < P_b \\ L\left(\sqrt{P_b} - \sqrt{P_a}\right) p_y & \text{if } P \ge P_b. \end{cases} \tag{12}$$

This formulation captures the non-linear and price-dependent nature of a concentrated liquidity position's dollar value. When inside the range, the position value depends on the prices of tokens, their ratio, as well as the chosen range. Outside the range, the position effectively behaves as a spot position in either token x or token y, depending on the direction of the price movement. The dollar value changes of a concentrated liquidity position, when the price of a single token changes (in this example p_x), are presented in Figure 4. The plot also shows the value of a passive 50/50 portfolio of the two tokens x and y. The highlighted areas indicate the active ranges of the three different liquidity positions. To isolate the effects of changes in the relative price of the tokens and the impact of price range selection, p_y was kept constant in the figure, even though depending on the pool it can fluctuate as well. The Figure 4 shows that below the active price range the value of the position decreases linearly with p_x, reflecting the fact that below the active price range the position is entirely held in token x, the underperforming asset. At the same time the liquidity position stops earning fees, when the price moves outside the active range. Conversely, above the active range the position value is flat, indicating full exposure to token y, the less appreciating asset.

When the values of the liquidity positions are compared to that of the passive 50/50 portfolio, it becomes evident that the deviation between the two increases as the price moves further away from the center of the active ranges (in this case $p_x = 1$). However, the value of widest position is close to the passive portfolio value when $0.9 \leq p_x \leq 1.1$, indicating greater robustness to price fluctuations near the midpoint. Since all positions are constructed to be 50/50 at the midpoint and normalized to equal value at that price, the difference between the passive portfolio value and the liquidity position value directly corresponds to the effect of deploying capital into a liquidity pool instead of holding the tokens passively. This difference is known as impermanent loss.

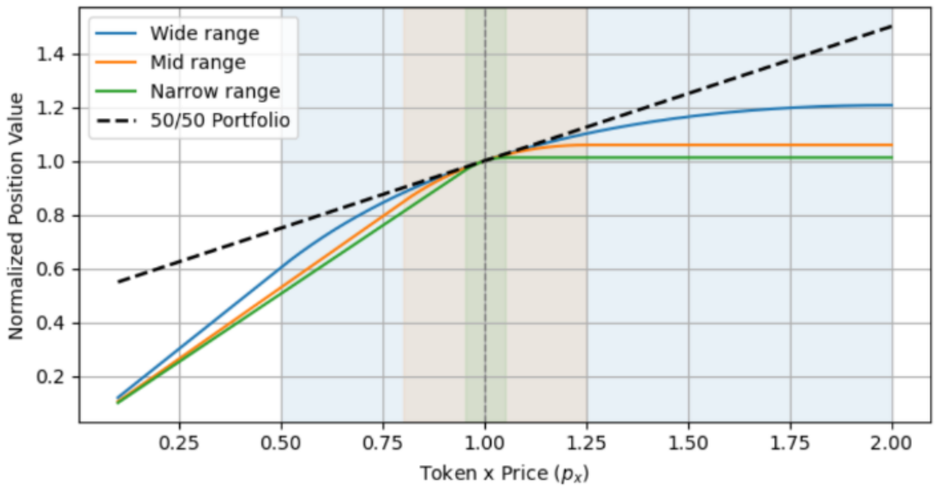

Figure 4: Normalized dollar values of three concentrated liquidity positions with different active ranges as a function of token x price p_x. The value of a passive 50/50 portfolio of tokens x and y is also shown for comparison. Highlighted areas mark the active ranges for each position. The dollar values are normalized at the middle of the active range for each liquidity position. Token y price p_y is kept constant to highlight the change in concentrated liquidity position value when the price of single token moves and the effect of price range in the value change.

Concentrated liquidity provision exposes LPs to several risks and challenges, primarily stemming from the non-linear behavior of the position's value. The most significant risk from the LP's perspective is a loss in value denominated in fiat currency (e.g., USD or EUR). As demonstrated previously, an unhedged liquidity position tends to lose value more rapidly than the average of the underlying tokens when their prices decline. This effect is amplified when the price deviates further from the midpoint of the position's active range, meaning that volatility expansion in a downtrending market represents a worst-case scenario for an unhedged LP.

On the upside, while there is no risk of nominal loss in fiat terms, the value of the position grows more slowly than the average of the underlying tokens. This asymmetric payoff profile creates a significant challenge for hedging. Consider an LP who passively hedges their exposure by shorting a 50/50 portfolio of the underlying tokens. In this case, a sudden increase in the price of one token can cause the liquidity position to underperform relative to the hedge, resulting in losses due to over-hedging. Conversely, during price declines, the hedge may not be sufficient to offset the accelerated loss in LP value, leading to under-hedging losses.

Therefore, effective hedge management must strike a delicate balance: it must avoid over-hedging during bullish scenarios while still providing adequate protection in bearish and volatile conditions. Static hedging strategies often fail to adapt to the non-linear and path-dependent nature of LP value, highlighting the need for dynamic and adaptive hedging approaches.

While providing liquidity generates fee revenue that is distributed to LPs, this revenue is typically separate from the liquidity position itself. It does not alter the composition or valuation of the underlying position unless it is manually claimed and reinvested (compounded) into the pool. Moreover, fee accrual depends primarily on price volatility and the chosen liquidity range, rather than the hedging strategy or the value of the position. As such, fee revenue can be modeled as an additive cash flow that does not directly influence the hedging dynamics of the liquidity position and thus fee revenue is excluded from the analysis in this thesis.

One more important dynamic of concentrated liquidity positions is that they are inherently path dependent, meaning that their final value does not depend solely on the initial and final prices of the assets, but also on the trajectory taken by the price within the active range. This is because the composition of the LP position — the proportion of each token held — evolves continuously with price changes. For example, two paths that begin and end at the same price may lead to different final token allocations and thus different position values, depending on how much time was spent near the boundaries of the active range or whether the price exited and later re-entered the range. However, when modeling the value of a single, static liquidity position and excluding fee revenue, the position is path independent: its ending value depends solely on the final prices of the underlying tokens relative to the position's active range, regardless of the price trajectory taken to reach that point.

3.5 Hedging of Liquidity Positions

While concentrated liquidity provision can offer attractive fee income, it also exposes LPs to significant directional risk due to the price dynamics of the underlying assets. As discussed previously, concentrated liquidity positions exhibit nonlinear exposure to price movements and are effectively short volatility: while moderate price movements within the active range generate fee income, significant deviations outside the range lead to an unfavorable shift in asset composition and impermanent loss. When considering a single concentrated liquidity position, impermanent loss reflects underperformance relative to holding the initial amounts of the two tokens. However, if the price of one or both tokens increases, the fiat-denominated value of the position also rises, and the LP continues to earn fee revenue while the position remains active. In such cases, although the LP may underperform a passive holding strategy, the overall outcome is still positive in fiat terms. Thus, during upward price movements, the worst-case scenario for an LP is typically limited to relative underperformance rather than an absolute loss. In turn, if the price of one or both tokens decreases, the impermanent loss represents larger fiat-value losses for the LP compared to holding the tokens.

The goal of hedging for an LP is to reduce downside losses in fiat terms while maintaining upside potential. In other words, a prudent LP aims to protect against large drawdowns (decreases of dollar value of their position) without giving up gains when prices rise. Impermanent loss highlights why a passive hedge, such as shorting equal amounts of the underlying tokens, may work when prices are stable, but tends to perform poorly when prices trend strongly in either direction. Thus, to successfully hedge concentrated liquidity positions the hedging position needs active rebalancing.

Various hedging approaches have been proposed in recent literature. The goal of these strategies is to mitigate downside variance, while maintaining exposure to fee revenue. Mainly, the hedging of LP risk is done by constructing a replication portfolio that inversely mimics the dollar value changes of the LP position. Since LP position value changes nonlinearly with respect to the price of the underlying asset pair, such replication often involves using derivatives like options or futures. In principle, if the replication is successful, then a short position in this portfolio would offset the losses incurred from holding the LP position during adverse price movements.

Broadly speaking, hedging methods can be grouped into two categories: those using options or static replication techniques, and those using futures and dynamic strategies. An example of using options and static replication is [28], where the authors introduce a method to hedge liquidity positions by constructing a static option portfolio that minimizes the Profit and Loss (PnL) residual of the LP position across the entire price range. They derive the LP's PnL function as a function of final asset price and then leverage least squares regression to determine the optimal mix of call and put options. This one-off portfolio effectively offsets directional exposure without requiring continuous rebalancing, making it a practical static hedge, provided a sufficiently liquid options market exists.

A complementary static approach for hedging IL of liquidity positions is presented by Lipton et al. [27], who frame IL as a contingent claim with a nonlinear payoff, leading to the introduction of an "IL protection claim", which essentially is just a derivative that delivers the negative of IL at a specified maturity date. They propose two main methods for valuing and hedging this claim. First, they develop a static, model-independent replication strategy, which shows that the IL protection claim can be perfectly hedged using a portfolio of European vanilla options. This approach generalizes existing replication methods and assumes the presence of a sufficiently liquid options market. Second, they present a dynamic, model-based approach for valuing and hedging the IL protection claim under a risk-neutral measure. Using a broad class of asset price models with known characteristic functions, such as Black-Scholes and Variance Gamma models, they derive closed-form solutions for pricing and enable dynamic rebalancing of the hedge over time. These results demonstrate that, in markets with developed derivatives infrastructure, IL exposure can be managed effectively using traditional option pricing techniques.

An important contribution to the evaluation of dynamic LP strategies is the Loss-versus-rebalancing (LVR) framework introduced by Milionis et al.[31]. They formalize the concept of replication error between a liquidity position and a portfolio that continuously rebalances to replicate the position's exposures. In their framework, the LP position is viewed as a synthetic derivative with a nonlinear, path-dependent payoff structure. The central insight is that the LP position can be approximated at each moment by a replicating portfolio of the two underlying tokens (ETH and BTC, for example), adjusted dynamically as price changes. However, because the liquidity position's token composition changes continuously with market prices while the replicating portfolio is rebalanced only at discrete points in time, their values can deviate from one another.

This deviation is quantified by the Loss versus Rebalancing metric, which measures the cumulative difference in value between the liquidity provider position and the dynamic replicating portfolio. Mathematically, if V_t^{LP} denotes the value of the LP position at time t, and V_t^{repl} denotes the value of the self-financing replicating portfolio (rebalanced continuously), then

$$\mathrm{LVR}(t) = V_t^{\mathrm{repl}} - V_t^{\mathrm{LP}}.$$

A positive LVR implies that the LP underperforms the replicating strategy, which can be interpreted as the cost of providing liquidity. Importantly, LVR is zero at inception since the positions are equivalent, and accumulates over time as price moves cause the LP composition to diverge from the replicating portfolio. The authors also show that, in the absence of fee revenue, LP positions incur negative returns compared to dynamic replication, especially in volatile markets.

An example of futures based dynamic hedging strategy is presented by Zhang et al. [8], who propose a reinforcement learning-based framework for managing Uniswap v3 LP positions. The core objective in their setup is to minimize LVR, which in the paper is defined as the difference in value between a continuously rebalanced replication portfolio composed of perpetual futures and a static liquidity position. This loss represents the cost of hedging the LP position, as it measures the performance gap between remaining unhedged and maintaining continuous exposure replication. Zhang et al. formulate the hedging problem as an MDP, where the RL agent observes a state vector including price, fee revenue, range width and other relevant market indicators. The agent then selects two actions at each step: the width of the liquidity range to be deployed in the AMM, and the size of the short futures position to hedge directional exposure. By using PPO algorithm, the agent learns a policy that dynamically adjusts both the LP configuration and hedge in response to market changes. Their results show that the learned policy significantly outperforms baseline strategies that use fixed-width LP ranges or naive hedging, particularly during trending or volatile market conditions. Importantly, the approach does not rely on any strong assumptions about the distribution of asset prices or market dynamics, making it flexible and adaptive. This paper is particularly relevant for this thesis, as it shares the central objective of using deep reinforcement learning to hedge the directional exposure of LP positions using futures.

3.6 Option Hedging

Since there are very few studies on dynamic hedging of liquidity positions, we also examine literature on option hedging and replication. This is motivated by the structural similarity between the two problems: both involve managing nonlinear and time-varying exposures that require dynamic rebalancing under market frictions. Options are not the main focus of this thesis; therefore, the discussion of similarities between options and liquidity provision is presented at a relatively high level. The comparison primarily revolves around the concept of exposure to the underlying asset, as quantified by the option's delta.

An option is a financial contract that gives the holder the right, but not the obligation, to buy or sell an underlying asset at a predetermined price before or at a specified date. Delta is a fundamental concept that quantifies the sensitivity of an option's price to changes in the price of the underlying asset. Mathematically, the delta Δ of an option is defined as the partial derivative of the option price V with respect to the underlying asset price S, i.e.,

$$\Delta = \frac{\partial V}{\partial S}.$$

This quantity represents the instantaneous rate of change of the option's value as the underlying price moves. In simplified terms, delta can be interpreted as the exposure to the underlying asset. Holding an option with a delta of Δ is approximately equivalent to holding Δ units of the underlying asset. So for example an option with $\Delta = 0.5$ behaves like holding 0.5 units of the asset.

The delta of an option changes over time as the underlying price S, time to maturity, and implied volatility vary. This sensitivity of delta to changes in the underlying price is captured by the gamma of the option

$$\Gamma = \frac{\partial^2 V}{\partial S^2} = \frac{\partial \Delta}{\partial S}.$$

This means that any change in the price of the underlying asset leads to a change in the delta. Consequently, the exposure of an option position changes non-linearly. This behavior is analogous to the liquidity provider's position, where the changes in option delta correspond to changes in the token amounts held within the liquidity position. Therefore, hedging an option is conceptually similar to hedging a liquidity position, as both involve managing constantly changing exposure as the underlying price moves.

Option replication in the presence of proportional transaction costs, where perfect delta hedging becomes infeasible is first studied by Leland [10]. He proposes a discrete-time hedging strategy in which the hedge is rebalanced at fixed time intervals and each transaction incurs a cost proportional to the volume traded. This leads to a modified option pricing formula that inflates the original volatility in proportion to the cost parameter and the hedging frequency, thereby embedding the cost of replication directly into the option's theoretical value.

Leland's method offers a tractable analytical solution for valuing and hedging contingent claims in markets with frictions, and his adjustment remains a cornerstone for understanding the interplay between rebalancing frequency and trading cost. Although originally designed for options, the approach is also conceptually relevant in the context of liquidity provision, as the framework highlights how infrequent rebalancing or high trading costs can lead to performance divergence between the LP position and its replicating portfolio. This insight is particularly relevant when evaluating the cost-effectiveness of hedging strategies.

One of the earliest RL implementations on options hedging is proposed by Halperin [36], called the QLBS model. QLBS is an that applies Q-learning to discrete-time option hedging within the Black-Scholes-Merton framework. The main goal in [36] is to show how dynamic hedging of derivatives can be framed as an MDP, enabling the use of model-free reinforcement learning instead of classical PDE-based approaches. In this model, the agent represents a trader who seeks to optimally hedge a short option position over a finite time horizon by trading the underlying asset. The underlying price follows a geometric Brownian motion, and trading occurs in discrete time steps.

The state space is defined by the current time step and the price of the underlying asset. The action space corresponds to the proportion of the portfolio allocated to the risky asset at each time step. The goal of the agent is to maximize expected terminal wealth minus a risk penalty, resulting in a quadratic utility objective. The reward function at each time step is designed as the negative of the variance of the hedged portfolio, effectively encouraging strategies that minimize risk.

The QLBS model constructs a backward recursion for the Q-function using Bellman optimality, in which the Q-value represents the expected cumulative utility (or cost) given a state-action pair. Importantly, the paper demonstrates how the Q-function can be estimated from data using linear regression, making the approach data-driven and model-free in discrete time, despite being inspired by a model-based continuous-time setting.

Halperin [36] does not include transaction costs, slippage, or liquidity constraints, consistent with assumptions from the classical BSM framework. Nevertheless, the QLBS model represents one of the earliest examples of using off-policy reinforcement learning for hedging, providing a foundational formulation for later research involving more realistic market conditions.

In the Deep Hedging framework proposed by Buehler et al. [16], the hedging problem is formulated as a sequential decision-making task where a neural network learns to produce hedge positions over time. Although they do not adopt classical reinforcement learning algorithms like Q-learning or policy gradient methods, it solves a reinforcement learning-style problem using supervised learning techniques. The state space includes observable variables such as the current price of the underlying assets, time to maturity, and features like historical prices or past hedge positions. The action space consists of the rebalancing decisions, i.e., the portfolio weights or changes in asset positions, taken at each time step. A feedforward neural network is trained to map the current state to the optimal hedge action. Rather than optimizing for immediate rewards, the network is trained to minimize a global objective function, typically a convex risk measure such as the expected shortfall or entropic risk of the terminal hedging error. The training process relies on simulated price trajectories, generated under models like Black–Scholes or Heston, and uses backpropagation through time to adjust the network parameters. While the simulated data assumes a model, the hedging strategy itself is learned in a model-free manner, without requiring explicit knowledge of the price dynamics. This architecture allows the agent to learn complex hedging strategies in high-dimensional, incomplete markets.

Another RL framework for dynamic replication and hedging of derivative instruments in discrete time is presented by Ritter and Kolm [37], whose approach explicitly formulates the hedging problem as an MDP, allowing the use of model-free RL algorithms to learn optimal hedging strategies directly from data. The state space includes features such as the time remaining to maturity, the current price of the underlying asset and existing portfolio positions. The action space corresponds to the set of allowable changes in the hedge position at each time step. A key contribution in [37] is the use of Q-learning to estimate the optimal action-value function that maps each state-action pair to the expected cumulative reward. The reward function reflects hedging performance and is a combination of transaction costs and hedging error. To train and evaluate the RL agent, Ritter and Kolm generate synthetic data from geometric Brownian motion models, and also test the algorithm on real financial data. Their results show that the learned strategies can replicate the payoff of derivatives effectively, even in the presence of market frictions such as transaction costs.

Building on [37], Du et al. [9] propose a model-free DRL approach to option replication and hedging that explicitly accounts for market frictions such as transaction costs and model uncertainty. They formulate the hedging problem as an MDP, where the state space includes variables such as the underlying asset price, time to maturity and current portfolio holdings. The actions correspond to adjustments in the hedging position. The hedging policy is parameterized by a deep neural network and trained using three methods: Q-learning network (DQN), DQN with Pop-Art normalization and PPO, enabling the agent to learn adaptive strategies that optimally balance the replication accuracy and trading costs in a purely data-driven manner, without relying on a specific stochastic model of asset price dynamics. A key insight from the study is that the performance of model-free DRL methods are often improved when historical data is used with simulated data. Thus, one important aspect of model-free DRL methods is simulating realistic and high quality data for the training. In their simulations DRL-based hedge outperforms traditional delta-hedging strategies, particularly in settings with high transaction costs or model misspecification. PPO performed the best out of the tested models in terms of delta-neutrality, training time and the amount of data required.

A model-free deep reinforcement learning approach using Deep Deterministic Policy Gradient (DDPG) for hedging portfolios of European options under realistic market frictions is proposed by Cao et al. [15]. DDPG is an actor-critic algorithm designed for continuous state and action spaces. Unlike the previously discussed papers, they use continuous state action space to avoid discretization error. The state space consists of features such as underlying asset prices, option Greeks (delta,gamma etc.), time to maturity and current hedging positions, and the agent outputs hedge adjustments at each time step. The reward function penalizes large hedging errors and transaction costs, guiding the agent toward risk-efficient and cost-aware strategies. The simulated training data is generated using a multi-asset stochastic volatility model

(Heston), enabling controlled experiments under realistic dynamics. Their results show that the DDPG-based agent outperforms traditional delta-hedging methods, especially in multi asset settings with significant frictions.

4 Problem Formulation

To train a reinforcement learning agent to manage the risk of a concentrated liquidity position (such as Uniswap v3), we implement a dedicated hedging environment based on historical price data. The environment simulates the evolution of an LP position and a hedge account composed of token futures. At each timestep, the agent observes a state comprising variables related to the LP dynamics, market trends and momentum indicators (such as volatility and moving averages), and the current hedge allocation. Based on this information, the agent selects actions that adjust the hedge, which are executed while accounting for transaction costs and portfolio constraints. The environment then updates the portfolio value and provides a reward signal designed to promote desirable hedging behavior.

4.1 Problem Setup

The primary objective of this thesis is to construct a dynamic hedging strategy that minimizes the downside risk associated with providing concentrated liquidity in a Uniswap v3 pool. In particular, we aim to reduce the fiat-denominated variance of the LP's portfolio value when prices move adversely, while allowing the LP to retain some upside potential when asset prices rise.

A naive approach would be to fully replicate the liquidity position through continuous rebalancing. However, this results in loss-versus-rebalancing (LVR) [31] in both uptrends and downtrends, since the hedger is forced to sell in rising markets and buy in falling markets. Instead, our goal is to design a hedge that functions more as a form of insurance against large losses rather than as a strict replication strategy. The hedge should absorb downside volatility while allowing some exposure in trending markets.

To achieve this, the hedging strategy relies on positions in futures contracts on the underlying tokens to manage risk exposure arising from the liquidity provision position. Futures contracts are standardized financial instruments that derive their value from an underlying asset and obligate the buyer and seller to transact at a predetermined price at a specified future date. In traditional financial markets, futures contracts are widely used both for speculation and for hedging, since they allow market participants to transfer price risk without exchanging the underlying asset immediately. In cryptocurrency markets, the most prevalent form is the perpetual futures contract. Unlike conventional futures, perpetual contracts have no expiry date, enabling positions to be held indefinitely. Their prices are kept close to spot prices through a funding rate mechanism, under which long and short traders exchange periodic payments. These payments, which typically occur at fixed intervals (e.g., every 8 hours), serve to align the perpetual futures price with the spot price. When the perpetual trades above the spot price, traders holding long positions pay those holding short positions; conversely, when the perpetual trades below the spot price, short positions pay long positions. This mechanism incentivizes market participants to arbitrage away deviations, keeping the futures price closely tethered to the underlying asset.

From a hedging perspective, perpetual futures are particularly useful because gains and losses on the contract can be chosen to move inversely to the price of the underlying asset. For example, a liquidity provider who is exposed to a decline in the value of ETH through their Uniswap v3 position can open a short ETH perpetual futures contract. If the spot price of ETH falls, the LP position loses value, but the short futures position generates a compensating profit, reducing overall portfolio variance. Conversely, if ETH appreciates, the LP's futures position incurs a loss, but this is offset by the increased value of the LP position. In this way, perpetual futures allow liquidity providers to scale and balance their exposure dynamically, creating an effective risk management tool.

Within this framework, the hedge is parameterized by hedge ratios, which determine the proportion of each token's exposure in the LP that is hedged via futures. These ratios allow the agent to adjust the hedge dynamically in response to market conditions, offering flexibility to mitigate losses while retaining partial exposure to potential gains. To avoid excessive rebalancing, the hedge can be updated at discrete intervals.

From a reinforcement learning perspective, the problem is framed as a sequential decision-making task. At each timestep, the agent observes state information related to the LP composition, market conditions, and current hedge allocation. Based on this information, the agent selects actions in the form of hedge adjustments, which are executed by the environment. The agent receives a reward based on the resulting hedge-adjusted portfolio performance, which guides learning toward strategies that reduce downside risk while avoiding over-hedging.

4.2 Simulation Environment

The simulation environment maintains the state of the agent's liquidity provision and hedging positions, including the current token balances within the Uniswap v3 pool, the total fiat-denominated value of the LP position, and the outstanding hedge positions in futures contracts. The environment provides an interface for executing hedge adjustments at discrete time intervals, corresponding to the 15-minute frequency of the price data.

4.2.1 Overview of Environment Dynamics

The simulation environment is designed to model the joint evolution of a Uniswap v3 liquidity provision (LP) position and a corresponding hedge account composed of perpetual futures contracts. The primary objective of the environment is to provide a controlled setting for training reinforcement learning agents to dynamically manage hedge positions while observing the resulting portfolio evolution. The LP position represents liquidity provided within a specified price interval $[P_a, P_B]$ in a Uniswap v3 pool. At each discrete timestep, the environment calculates the token amounts held within the pool based on the current spot price, the specified liquidity, and the predetermined bounds of the pool. The LP position evolves passively according to the Equation (9) and is not actively modified by the agent. Consequently, the fiat-denominated value of the LP depends solely on the token composition and prevailing market prices based on Equation (12).

The hedge account consists of positions in perpetual futures contracts on the underlying tokens. These positions are modeled as self-financing, such that realized gains and losses are incorporated directly into the hedge account balance without external replenishment. Losses in the hedge do not affect the LP position, and the simulation continues under the assumption that the hedge positions are continuously financed. In practice, the PnL of the hedge and the PnL of the LP position are computed separately at each timestep. The total portfolio PnL is then obtained as the sum of these two components, reflecting the combined performance of the LP and the hedge positions. The target hedge ratios for each token are determined by the agent, and the environment executes the corresponding futures trades, accounting for transaction costs and discrete rebalancing constraints. By maintaining a strict separation between the LP and hedge positions, the environment ensures that the LP evolves independently while the hedge is actively managed according to agent decisions, enabling an accurate assessment of the hedge's effectiveness in mitigating downside risk.

At each timestep, the environment provides the agent with a state vector containing variables such as normalized token prices, LP values, current hedge positions, and market indicators. The agent selects target hedge ratios, which the environment translates into futures trades executed immediately. Any changes in the hedge positions incur transaction costs, and the hedge account balance is updated to reflect realized profits and losses. Simultaneously, the LP value is recomputed based on the updated market prices. The total portfolio value, comprising both the LP and hedge components, is then calculated and recorded for reward computation and subsequent analysis. This structure enables the environment to simulate the evolution of a concentrated liquidity position under realistic market conditions, while providing the agent with sufficient information to make informed hedging decisions.

4.2.2 Data

The simulation environment requires financial time series that represent the underlying assets of the concentrated liquidity position and the instruments available for hedging. In the base configuration, the pool consists of two volatile cryptocurrencies, denoted token i and token j. This choice reflects the fact that many Uniswap v3 pools are formed from pairs of major digital assets such as ETH and BTC, where both assets exhibit significant and often correlated price dynamics. Nonetheless, the framework is not restricted to this setting. With minor modifications, the framework can also accommodate pools involving a volatile cryptocurrency paired with a stablecoin pegged to USD, which are among the most common and economically important trading pairs in decentralized exchanges.

For this thesis, the tokens are instantiated as ETH and BTC. The corresponding spot price series are denominated in USD and retrieved from the ETH/USDT and BTC/USDT trading pairs on the Binance exchange. USDT (Tether) is a stablecoin designed to maintain 1:1 peg with the US dollar, providing a reliable reference for pricing cryptocurrencies in fiat terms. The ETH/USDT and BTC/USDT pairs were chosen specifically because they provide the longest and most complete historical data series, out of any ETH-stablecoin pairs, enabling multi-year analyses at 15-minute intervals. Binance is the largest cryptocurrency exchange by trading volume and provides deep liquidity in these major pairs, making its spot prices a reliable proxy for the prevailing market valuations of the underlying assets. Specifically, the close prices of the 15-minute OHLC (open,high,low,close) price data beginning on 1.1.2021 UTC and ending on 31.8.2025 UTC from Binance's public API was used.

Furthermore, the ETH/BTC price series is constructed directly from the ETH/USDT and BTC/USDT data by taking the ratio of their respective prices. Formally, at each time step t, the ETH/BTC price is computed as:

$$\text{ETH/BTC}_t = \frac{\text{ETH/USDT}_t}{\text{BTC/USDT}_t}$$

This approach aligns all three series in time and avoids inconsistencies that could arise from using independently sourced ETH/BTC data, thereby maintaining a realistic representation of relative price movements between the two assets.

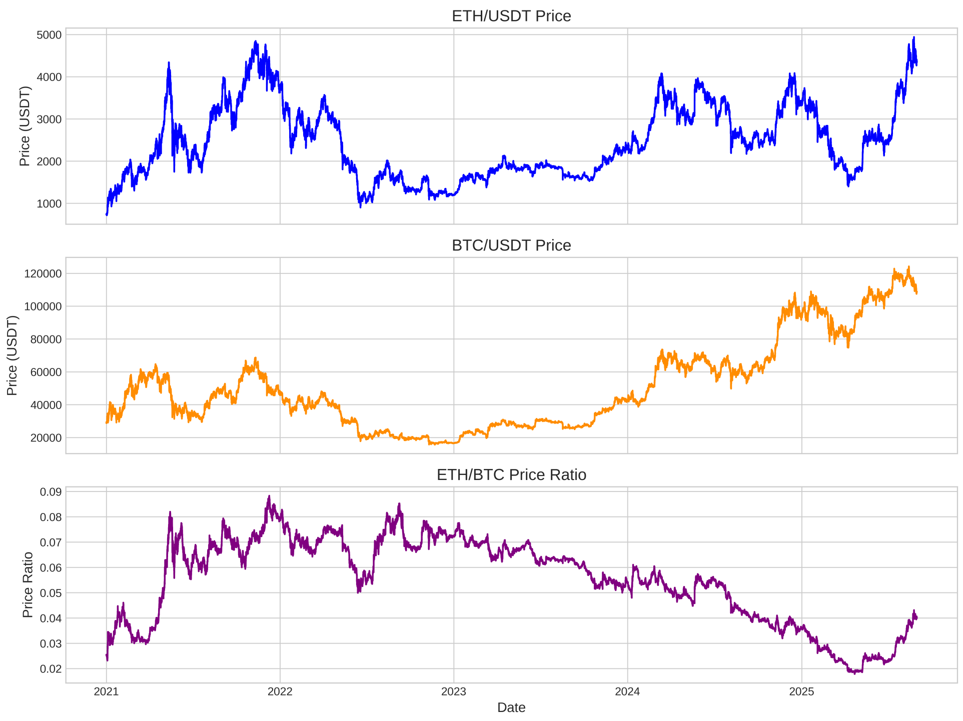

Figure 5: Historical price series for ETH/USDT, BTC/USDT, and the ETH/BTC ratio. The data are sampled at 15-minute intervals from 1.1.2021 to 31.8.2025.

Figure 5 presents the price data used in this thesis. The time series are characterized by high volatility, with pronounced trends and abrupt price movements that are typical of major cryptocurrency markets. Sampling at 15-minute intervals ensures that short-term dynamics are captured while avoiding excessive noise that might arise from ultra-high-frequency tick data. These series form the basis for computing portfolio valuations, hedge exposures, and ultimately the state space observations for the reinforcement learning agent.

The funding rate data used in this thesis are obtained from Binance's perpetual futures markets for the ETH/USDT and BTC/USDT pairs. Funding rates represent the periodic payments exchanged between long and short positions in perpetual futures, designed to keep the contract price close to the spot market. The raw data from Binance consist of historical funding events, which occur every eight hours, with each record specifying the timestamp and the corresponding funding rate. Because the simulation environment operates at 15-minute intervals, the raw funding rate series is resampled to this frequency using forward-filling, ensuring that each 15-minute timestep is associated with the most recently observed funding rate. The resulting series provides a continuous, 15-minute funding rate input that is temporally synchronized with the 15-minute spot price data used for ETH and BTC.

Formally, if f_t denotes the funding rate at the most recent eight-hour funding event prior to time t, then the resampled 15-minute funding rate series assigns f_t to all 15-minute intervals following that event until the next observed funding timestamp. This allows the reinforcement learning agent to account for the effect of funding payments on hedge positions while maintaining temporal consistency with the underlying price series.

Together, the spot prices and funding rate data provide the necessary input to model the dynamics of the concentrated liquidity position and to try and implement a hedging strategy that mitigates downside risk while preserving exposure to upside movements.

4.2.3 State Space

In reinforcement learning, the design of the state space is central to how effectively an agent can interact with and learn from its environment. The state space must provide a compact yet sufficiently informative representation of the system dynamics so that the agent can identify profitable strategies. In the context of managing a Uniswap v3 liquidity provision position, the state space is constructed to capture three distinct but complementary aspects of the problem: the liquidity provision itself, the surrounding market conditions, and the current hedging stance of the portfolio. Together, these features aim to summarize both the immediate risks and the opportunities available to the agent.

A total of 20 variables are included in the observation vector at each time step. All variables are normalized in order to reduce non-stationarity across training episodes that begin at different points in time, and to stabilize the learning process of the policy network. For example, raw token prices are divided by their initial values, and portfolio-related quantities such as LP and hedge values are scaled by the initial portfolio value. Volatility measures are clipped and rescaled to limit the impact of extreme fluctuations. This normalization ensures that the agent learns on a consistent feature scale, independent of absolute market levels.

To provide a clear structure, the state variables are grouped into three categories. The first group consists of liquidity provision variables, which provide the agent information about the held LP position. The second group consists of variables about underlying market dynamics and short- versus long-term price behavior. The final group consists of hedge position variables, which describe the agent's existing hedging stance and the funding costs associated with maintaining it. This grouping highlights three core aspects of the problem, the liquidity position, the prevailing market environment and the current hedge position.

The liquidity provision variables describe the current state of the LP position within the Uniswap v3 pool, focusing on the agent's exposure rather than market signals. This group is presented in Table 1. The group includes the normalized values of Token1 and Token2 held in the pool, expressed relative to the initial portfolio value at the start of the episode. Since the prices of the tokens fluctuate, these variables inform the agent about how the position value evolves as well as presenting information about the composition of the LP position. By scaling the token values relative to the initial portfolio, the agent receives a consistent measure of exposure across episodes with different starting conditions.

Next the normalized price ratio of the underlying tokens is defined as an LP variable. The ratio price is normalized by dividing its current value by the initial value at the start of the episode. This transformation expresses movements as percentage changes relative to the baseline rather than as absolute levels. In doing so, the variable retains information about the magnitude and direction of price movements, while abstracting away from absolute price levels that may otherwise introduce distortions. This ratio is critical because it directly determines the composition of the LP position; changes in the ratio alter the amounts of Token1 and Token2 held in the pool. By observing the ratio price, the agent can infer how the token allocation evolves over time, which is essential for assessing exposure and designing an effective hedging strategy.

In addition, the normalized width of the LP range is included as a state variable. This width captures how concentrated the liquidity position is and reflects the sensitivity of the token composition to price movements. Narrower ranges produce larger changes in token allocation for a given price change, potentially requiring more precise hedging. The width of the LP range informs the agent about the sensitivity of the current token composition to changes in the token prices. Collectively, the LP variables provide the agent with a clear view of the position's current composition and associated risk profile, while keeping the observation vector concise. To maintain simplicity in the state space, only LP variables that were expected to provide meaningful, non-redundant information were included.

The second group is presented in Table 2. The market dynamic variables capture information about short- and long-term price behavior that is not directly observable from the LP state alone. These features aim to provide the agent with signals about momentum, trend, and volatility, helping it anticipate how the liquidity position might evolve under different market conditions.

The individual token prices are included in this group to capture overall market direction and to indicate how changes in each asset contribute to movements in the ratio price. As with the ratio price, these variables are normalized by dividing by their initial values, so that only the relative magnitude and direction of price changes are preserved. This ensures that the agent focuses on proportional movements rather than absolute price levels.

Table 1: Liquidity pool (LP) variables included in the PPO observation vector. The normalized LP value formula applies to both Token i and Token j.

Variable	**Formula**	**Description**
$V_{\text{LP},k}^{\text{norm}}$	$\dfrac{q_k \cdot P_k}{V_{\text{LP},0}}, \quad k \in \{i, j\}$	Value of Token k (either token i or token j) in the LP, normalized by initial total LP value $V_{\text{LP},0}$.
P_{ij}^{norm}	$\dfrac{P_{ij}}{P_{ij,0}}, \quad P_{ij} \equiv \dfrac{P_i}{P_j}$	Normalized ratio price of Tokens i, j.
W^{norm}	$\dfrac{\overline{P}_{ij} - \underline{P}_{ij}}{P_{ij,0}}$	Normalized width of the LP range. Encodes how concentrated the liquidity position is and indicates sensitivity of token composition to price changes.

To capture medium- and longer-term price dynamics, the observation space includes exponential moving averages (EMAs) of both the individual token prices and the ratio price. Each EMA is expressed as a relative deviation from the current price, so that values close to zero indicate little divergence, while positive or negative deviations highlight upward or downward momentum. Two horizons are considered: a shorter 20-period EMA, which reflects more recent price trends, and a longer 100-period EMA, which captures more persistent directional shifts. Including EMAs for both the individual token prices and the ratio price provides the agent with a richer signal set, enabling it to recognize whether market movements are driven by one asset in particular or by shifts in their relative valuation.

The exponential moving average (EMA) of a price series P_t with period length n is defined recursively as

$$\text{EMA}_t = \begin{cases} P_0, & t = 0, \\ \alpha P_t + (1 - \alpha)\text{EMA}_{t-1}, & t = 1, 2, \dots, T, \end{cases}$$

where $\alpha \in (0, 1)$ is the smoothing parameter, chosen as $\alpha = \frac{2}{n+1}$, where n is the period length and t is the time index. This weighting scheme ensures that more recent prices have a higher influence, while past values decay exponentially in importance.

Table 2: Market dynamic variables included in the PPO observation vector. Normalized prices and EMA deviations include both Tokens i, j. The volatilities and EMA deviations of the ratio price P_{ij} are also included.

Variable	Formula	Description
P_k^{norm}	$\dfrac{P_k}{P_{k,0}}, \quad k \in \{i, j\}$	Token price normalized by its initial value; conveys market level relative to episode start.
σ_h	$\sigma_h = \sqrt{\dfrac{1}{h-1}\sum_{k=0}^{h-1}\left(\ln\dfrac{P_{ij,t-k}}{P_{ij,t-k-1}} - \bar{r}_h\right)^2}$ $\bar{r}_h = \dfrac{1}{h}\sum_{k=0}^{h-1}\ln\dfrac{P_{ij,t-k}}{P_{ij,t-k-1}}, \; h \in \{5, 20\}$	5-period and 20-period volatility of log returns of the ratio price P_{ij}.
$\text{EMA}_n^{(k)}$	$\dfrac{\text{EMA}_n(P_k)}{P_k} - 1, \quad k \in \{i, j\}, \; n \in \{20, 100\}$	Relative deviation of n-period EMA from Token k price; captures short- and long-term trends.
$\text{EMA}_n^{(ij)}$	$\dfrac{\text{EMA}_n(P_{ij})}{P_{ij}} - 1, \quad n \in \{20, 100\}$	Relative deviation of n-period EMA from the ratio price P_{ij}; captures short- and long-term relative trends.

Finally, two volatilities of the ratio price are included to provide the agent with a measure of market uncertainty and to help distinguish between minor fluctuations and movements that require active hedging. The volatilities considered are the 5- and 20-period standard deviations of the logarithmic returns of the ratio price over their respective periods. Only the ratio price volatility is considered, as it directly influences the rate at which the LP composition changes within the active range. The relatively short periods were chosen because large jumps are typical for cryptocurrency prices. For stability during training, the raw volatility values are clipped to a maximum of 0.1 and then scaled by dividing by 0.05. This normalization ensures that extreme market movements do not produce excessively large inputs, while keeping the magnitude of typical fluctuations within a consistent range that the agent can effectively learn from.

The final group of variables, presented in Table 3, captures the state of the hedge position. Analogous to the normalized LP token values, the normalized hedge token values are included in the observation vector. These variables describe the current size of the hedge position relative to the initial portfolio, providing the agent with a consistent metric for evaluating the hedge. In addition to the absolute hedge values, the state vector also includes the hedge ratios for each token, defined as the proportion of the corresponding token's LP amount that is currently hedged. These ratios provide the agent with a direct measure of how fully each component of the LP position is covered, independent of price changes. By observing both the normalized hedge values and the hedge ratios, the agent can assess not only the dollar exposure of its hedge but also its relative effectiveness in mitigating LP risk.

The final component of the hedge-related state variables is the funding rate, which reflects the cost or income from holding a leveraged hedge position on perpetual futures and can also signal overbought or oversold conditions. The funding rate is normalized similarly to the volatilities, clipped to a maximum absolute value of 0.01 and and then scaled to the $[-1, 1]$ range. This brings the funding rate onto a comparable numeric range as the ratio price volatilities, ensuring stable learning for the agent. By observing the funding rate, the agent can incorporate the cost of maintaining the hedge into its decision-making, balancing the benefits of risk reduction against the potential ongoing expense.

Table 3: Hedge variables included in the PPO observation vector.

Variable	**Formula**	**Description**
H_k	$\dfrac{h_k\,P_k}{V_{\mathrm{LP},0}}, \quad k \in \{i, j\}$	Normalized value of the Token k hedge; h_k is the hedged quantity of Token k.
$\dfrac{h_k}{q_k}$	$\dfrac{h_k}{q_k}, \quad k \in \{i, j\}$	Hedge ratio for Token k: fraction of LP exposure q_k that is hedged.
f_k^{scaled}	$f_k^{\mathrm{scaled}} = \dfrac{\mathrm{clip}(f_k, -0.01, 0.01)}{0.01}$ $k \in \{i, j\}$	Futures funding rate for Token k, clipped to $[-0.01, 0.01]$ and scaled to $[-1, 1]$.

The selection of these variables was guided by logical considerations regarding what information is relevant for effective hedging, rather than through a formal feature selection or statistical analysis, as that is outside the scope of this thesis. Notably, the overall portfolio PnL is not directly included in the state vector; instead, the agent observes the components of the portfolio, including LP values and hedge values, which implicitly contain information about the position's profitability.

Although the ratio price is directly derived from the individual token prices, it is retained as a separate variable because it has a distinct significance for the LP composition and volatility calculations. Collectively, the chosen state variables provide the agent with a structured and consistent representation of the position's current exposure, market conditions, and hedge status, facilitating informed decision-making while maintaining a compact and interpretable state space.

4.2.4 Action Space

The agent's actions specify adjustments to the hedge applied to the Uniswap v3 LP position. At each decision step the policy outputs a compact action vector that is interpreted as target hedge ratios for each token in the pool. The environment then converts these targets into executed trades, applies transaction costs, and updates the hedge and portfolio state. The discrete, ratio-based action parameterisation keeps decisions interpretable and limits the magnitude of single-step rebalancing, which helps control transaction costs and stabilise learning.

At each decision step t, the agent selects an action

$$a_t = \left(h_t^{(i)}, h_t^{(j)}\right),$$

where $h_t^{(i)}$ and $h_t^{(j)}$ denote the target hedge ratios for Token i and Token j exposures in the liquidity position, respectively. The hedge ratios are drawn from a finite discrete set,

$$h \in \{0.50, 0.55, 0.60, \ldots, 1.20\}.$$

A hedge ratio of $h = 1.0$ corresponds to a fully hedged exposure. Values below one represent partial hedging, where only a fraction of the exposure is neutralized, while values above one represent over-hedging, in which the hedge exceeds the underlying exposure and results in a net short position. Conceptually, if the liquidity position currently holds q_i units of Token i and q_j units of Token j, then the action determines how much of each exposure is hedged using futures or equivalent short positions. For example, the action $(0.50, 1.00)$ specifies that 50% of the Token i exposure and 100% of the Token j exposure should be hedged. Similarly, the action $(1.20, 0.80)$ corresponds to over-hedging Token i by 20% and hedging Token j by 80%. After the agent selects target hedge ratios, the environment compares them to the current hedge ratios and executes a rebalance only if the deviation exceeds 10% for a given token exposure. For instance, if the agent chooses target hedge ratios $(0.80, 0.90)$ and the current hedge ratios are $(0.82, 0.79)$, the Token i hedge is kept unchanged because the deviation $|0.82 - 0.80| = 0.02$ is below the 10% threshold, while the Token j hedge is rebalanced to 0.90 of the Token j exposure since the deviation $|0.79 - 0.90| = 0.11$ exceeds the tolerance band. This mechanism ensures that small differences between the current and target hedge ratios do not trigger unnecessary rebalancing or incur additional transaction costs.

A couple of design decisions were made in defining the action space and hedge representation. First, the hedge ratios are chosen from the previously shown discrete set rather than allowing continuous values. Using a discrete action space simplifies the reinforcement learning problem by reducing the total number of possible actions, which improves training stability and makes the resulting policy more interpretable. In contrast, a continuous action space would provide improved control over hedge ratios but would significantly increase exploration complexity, slow convergence, and potentially introduce unnecessary fluctuations in hedge adjustments that are small but still incur transaction costs.

Second, the hedge ratios are expressed as percentages of the LP token amounts rather than absolute contract sizes. This choice ensures that the agent's decisions are scale-invariant, meaning the same hedge ratio applies consistently regardless of the total size of the liquidity position. This also stabilizes the neural network training, since the network outputs and input features remain within a similar numeric range.

While the environment captures the essential mechanics of hedge management, several simplifications are made. First, the hedge ratios are capped between 50% and 120% of the underlying exposure, excluding extreme under-hedging or aggressive over-hedging that might occur in practice. This design choice prevents extreme under-hedging, which would leave the portfolio fully exposed, and extreme over-hedging, which would create excessively large directional positions. Capping the hedge ratios also improves training stability and ensures that all actions remain within a realistic and interpretable range for liquidity management. Second, hedge orders are assumed to be executed immediately at a price that is adjusted for proportional transaction costs, which provides a simplified approximation of execution frictions. More complex market microstructure effects, such as order book dynamics or varying liquidity, are not explicitly modeled. These simplifications allow the focus to remain on the interaction between liquidity provision and systematic hedge management, while providing a tractable setting for reinforcement learning.

4.2.5 Reward Function

In reinforcement learning, the reward function encodes the objective that the agent seeks to optimize. In the context of hedging a Uniswap v3 liquidity provision position, the reward must reflect the economic trade-off between mitigating the risk of underlying token prices decreasing, LP position differing from the hedge position and incurring transaction costs from frequent rebalancing. A well-designed reward provides the agent with a signal that balances these competing objectives and guides it toward policies that are both profitable and risk-aware.

Hedging a Uniswap v3 liquidity position induces LVR [31], reflecting the fact that continuously rebalancing the hedge incurs costs due to executing trades at less favorable prices than the changes in the LP position composition. Furthermore, in the absence of fee revenue, the LP position consistently underperforms a static portfolio composed of the initial liquidity allocation of the underlying tokens, as illustrated in Figure 4. Together, these two dynamics create a setting in which the expected outcome of holding and hedging the LP position is to end up with less than the starting value, due to the asymmetric nature of the problem. In practice, given two equal starting LP and hedge positions, an equally sized move in the underlying token prices to the upside generates a smaller increase in portfolio value (LP plus hedge) than an equivalent move to the downside decreases the portfolio value.

Another important dynamic of the problem is that minimizing the one-step variance would require rebalancing the hedge position to match the LP position whenever they differ sufficiently. Such a strategy would incur maximal LVR, resulting in a cost at nearly every step. Therefore, minimizing variance is likely to produce outcomes similar to a manual rebalancing strategy where the hedge is updated at every step. It is also worth noting that only minimizing downside variance provides any practical benefit from the liquidity provider's perspective.

In order to define the reward function, it is useful to consider the portfolio profit and loss (PnL) at each step. The total PnL can be decomposed into three components: the change in value of the LP position, the marked-to-market PnL of the hedge positions, and the transaction costs incurred from adjusting the hedge. Formally separating these components clarifies how each contributes to the overall portfolio dynamics and informs the agent's reward.

The value of the Uniswap v3 liquidity position at step t is determined by the current token prices and the composition of the LP position within its active price range. Let $q_{i,t}$ and $q_{j,t}$ denote the quantities of Token i and Token j in the LP position at step t, and let $p_{i,t}$ and $p_{j,t}$ denote their respective prices in USD. The LP position value is then

$$V_t^{LP} = q_{i,t} \cdot p_{i,t} + q_{j,t} \cdot p_{j,t}.$$

The change in LP value between two consecutive steps, i.e., the LP profit and loss (PnL), is given by

$$\Delta V_t^{LP} = V_t^{LP} - V_{t-1}^{LP},$$

The second part that affects the portfolio value is the hedge position. The hedge position PnL at step t represents the marked-to-market profit or loss from the hedging positions held to offset the LP exposure, including the effect of funding payments applied to the hedged positions. For most tokens on Binance, funding rates are paid at discrete times during the day: 00:00, 08:00, and 16:00 UTC. These payments are proportional to the value of the hedged positions at the funding time. Therefore, the funding contribution to the hedge PnL is calculated by multiplying the hedge quantity by the token price and the applicable funding rate. Thus the hedge position PnL at step t can be naturally decomposed into two components: the PnL from price movements and the PnL from funding costs. Let $h_{i,t-1}$ and $h_{j,t-1}$ denote the hedge quantities held during the previous step for Token i and Token j, and let $p_{i,t}$ and $p_{j,t}$ be the corresponding token prices at step t.

Then the marked-to-market profit or loss due to changes in token prices is

$$\Delta V_t^{\text{Price}} = -h_{i,t-1} \cdot (p_{i,t} - p_{i,t-1}) - h_{j,t-1} \cdot (p_{j,t} - p_{j,t-1}).$$

The other part of the hedging PnL comes from the funding payments. Funding payments occur only at discrete times (00:00, 08:00, and 16:00 UTC). The funding contribution to the hedge PnL is

$$\Delta V_t^{\text{Funding}} = h_{i,t-1} \cdot p_{i,t} \cdot f_{i,t} + h_{j,t-1} \cdot p_{j,t} \cdot f_{j,t},$$

where $f_{i,t}$ and $f_{j,t}$ are the funding rates applied at step t, non-zero only at the funding timestamps.

The total one-step hedge PnL is then simply the sum of the two components,

$$\Delta V_t^{\text{Hedge}} = \Delta V_t^{\text{Price}} + \Delta V_t^{\text{Funding}}.$$

This formulation ensures that the hedge PnL accounts both for the price changes of positions actually held in the previous step and for the realized effect of funding payments, providing an accurate measure of the hedging performance.

The transaction costs at step t are calculated based on the changes in the hedge positions. Let $h_{i,t}$ and $h_{j,t}$ denote the hedge quantities for Token i and Token j after rebalancing at step t, and let $p_{i,t}$ and $p_{j,t}$ be the corresponding token prices. The per-step transaction costs are defined as

$$C_t = c \cdot \Big(|h_{i,t} - h_{i,t-1}| \cdot p_{i,t} + |h_{j,t} - h_{j,t-1}| \cdot p_{j,t}\Big),$$

where c is the proportional transaction cost rate applied to each trade.

This formulation is chosen for several reasons. First, using a proportional cost reflects realistic trading fees and slippage in automated market makers and futures markets. Second, the absolute difference in hedge positions ensures that only actual changes incur a cost, so maintaining the same hedge does not generate unnecessary penalties. Since the portfolio value accounts for transaction costs, the agent effectively internalizes the cost of hedge rebalancing. Each adjustment of the hedge reduces the portfolio value by the corresponding trading costs, which naturally discourages unnecessary trades. As a result, the agent is incentivized to carefully consider when and how much to rebalance, balancing hedging effectiveness against trading costs.

Given the asymmetry of the LP hedging problem and the inherent cost of rebalancing (LVR), we choose as our reward the one-step portfolio return net of transaction costs. Let V_t denote the total portfolio value at step t, including both the LP position and any hedge positions, with transaction costs already subtracted. The reward R_t is then defined as

$$R_t = \kappa \cdot \frac{V_t - V_{t-1}}{V_{t-1}},$$

where κ is a scaling factor chosen for numerical stability during training.

This reward directly incentivizes actions that increase the portfolio value while accounting for the cost of hedging trades. Importantly, the asymmetric nature of the environment implies that downside losses are inherently more damaging than equivalent upside gains, so the agent is implicitly motivated to hedge against large losses without requiring an explicit variance penalty. Using this reward function, the agent can learn to balance the trade-off between reducing exposure to impermanent loss and minimizing transaction costs.

4.2.6 Episode Structure

At the beginning of each episode, the environment initializes the state of the liquidity provision and hedge positions. The starting time index is drawn randomly from the historical dataset, subject to the constraint that a full episode length can be simulated without exceeding the data horizon. This randomization ensures that the agent is exposed to a wide variety of market conditions during training, rather than overfitting to a particular segment of the data.

The Uniswap v3 liquidity position is initialized with a fixed notional amount of liquidity, distributed across a symmetric price range $[P_a, P_b]$ centered around the prevailing spot price at the episode start. The chosen width of the interval reflects the agent's passive exposure to relative price movements of the two tokens. The quantities of Token i and Token j allocated within the pool are computed from the initial spot prices according to the Uniswap v3 liquidity equations. Importantly, the LP position is not actively modified during the episode; it evolves passively as a function of the price trajectory within the specified range.

The hedge account is initialized such that both token exposures are fully hedged at the beginning of the episode, i.e. the initial hedge ratios satisfy

$$h_0^{(i)} = h_0^{(j)} = 1.0.$$

This way the portfolio starts from a neutral exposure to token price movements, allowing the agent's subsequent decisions to determine the evolution of risk and return. The hedge is implemented using perpetual futures contracts, which are modeled as self-financing instruments. Unlike the liquidity position, the hedge does not contribute to the initial portfolio value. Instead, the hedge is accounted for purely through the realized profits and losses it generates on a step-by-step basis once the episode has started. At time $t = 0$, the hedge account balance is therefore zero, and the portfolio is defined entirely by the LP position.

Formally, the initial portfolio value is given by

$$V_0 = V_0^{LP}, \qquad V_0^{Hedge} = 0.$$

The hedge is treated exclusively as a source of stepwise PnL rather than an asset with initial value. This modeling choice avoids unnecessary complexity from simulating cash transfers between LP and hedge accounts, which would not contribute to the main research question of comparing hedging strategies.

Finally, the environment constructs and returns the first observation vector, which encodes the normalized LP values, market indicators, and hedge state variables at time step $i = 0$. This observation forms the basis on which the reinforcement learning agent selects its first action in the subsequent step.

After the environment has been initialized, the episode proceeds in discrete steps, indexed by $n \in \mathbb{N}$, where the 0th step corresponds to the initialization. At the beginning of each step, the agent receives an observation vector that encodes the current state of the liquidity provision position, relevant market indicators, and the hedge account. Using this observation, the agent selects an action corresponding to the target hedge ratios for each token in the liquidity position.

The environment then translates the agent's action into trades in the perpetual futures markets. If the current hedge differs from the target hedge, a rebalance is executed, and transaction costs are applied. The hedge account is updated to reflect the realized profits or losses from the executed trades. In addition, funding payments, which occur every eight hours at 00:00, 08:00, and 16:00, are applied to the hedge positions, further affecting the stepwise PnL. Simultaneously, the LP position evolves passively according to the prevailing token prices and the specified Uniswap v3 range. Its fiat-denominated value is recalculated based on the updated token quantities and prices.

Following the updates to both the hedge and LP positions, the environment calculates the change in the LP value over the current step and adds the realized profit or loss from the hedge. Transaction costs from any hedge rebalancing are subtracted, yielding the total portfolio change. This quantity is then scaled to define the one-step reward for the agent.

Once the reward has been computed, the environment constructs the observation vector for the next step, incorporating the updated LP values, market indicators, and hedge state. At this point, the environment checks whether the episode termination condition has been reached. The episode ends once the cumulative number of steps exceeds the parameter rows_per_episode. An episode also terminates if the Uniswap v3 liquidity position moves out of its specified price range. An alternative modeling choice could allow the LP to recenter its range around the prevailing spot price, which would maintain the LP in the market and enable continuous learning of hedging strategies across a broader set of price trajectories. However, the termination-on-range-breach approach simplifies the environment and has little effect on training when episode lengths are close to the average time the price remains within the set ranges. If the termination condition is not satisfied, the cycle repeats, and the agent selects the next action based on the newly generated observation. The structure of a single step update is presented as a pseudocode in Algorithm 1.

Algorithm 1 Pseudocode of Step Update in the Uniswap v3 Hedging Environment

```
Require: Current state: LP values, hedge positions, market prices
Require: Agent action: target hedge ratios
Ensure: Next state, step reward, updated portfolio and hedge
 1: Observe current state
 2: Rebalance Hedge if needed
 3:     Adjust hedge positions
 4:     Apply transaction costs
 5: Update LP Composition and Value
 6:     Compute token quantities and LP value
 7: Calculate Hedge PnL
 8:     Determine profit or loss from hedge adjustments
 9: Update Portfolio Value
10:     Combine LP value and hedge PnL
11: Compute Step Reward
12: Advance Step
13: Check Termination Condition
14: Construct Next Observation Vector
```

5 Training Setup and Baselines

5.1 Data Split

The historical price and funding rate data used in this thesis were introduced in Subsection 4.2.2. For training and evaluation of the reinforcement learning agent, these data are divided chronologically into disjoint training, validation, and testing sets. This chronological split avoids lookahead bias and ensures that the agent is evaluated only on market conditions it has not seen during training.

The division is as follows:

- **Training set:** 1.1.2021 – 31.12.2023. This period spans three years and includes both bull and bear markets, providing the agent with a diverse set of market regimes for learning.
- **Validation set:** 1.1.2024 – 31.12.2024. This year is used exclusively for hyperparameter tuning and agent selection. Performance on this set informs early stopping and prevents overfitting to the training set.
- **Testing set:** 1.1.2025 – 31.8.2025. This held-out period is reserved for out-of-sample evaluation. The test period contains both upward and downward price movements, making it suitable for stress-testing the robustness of the hedging strategy.

The training, validation and testing split is visually presented in Figure 6. The ETH/USDT price data encompasses a variety of market regimes. Each of the training, validation, and testing splits contains at least one large upward movement as well as one large downward movement. This diversity is beneficial for agent generalization, as the agent is not trained or evaluated on only a single type of market condition. The BTC/USDT price data exhibits similar directional movements to ETH/USDT, although the magnitudes differ, and the validation and testing sets contain smaller drawdowns compared to ETH/USDT. The third price series, ETH/BTC, exhibits periods of both increase and decrease with varying volatility during the training set. In the validation set, however, the ETH/BTC ratio mostly trends downward with only a few small upward spikes, meaning the validation evaluation captures agent performance under this specific market regime. In contrast, the testing set includes both downtrends and uptrends in the ETH/BTC ratio, providing a broader assessment of the agents ability to generalize.

5.2 Training and Validation

The agents are trained using the Stable Baselines3 implementation of the Proximal Policy Optimization (PPO) algorithm [38], a model-free reinforcement learning algorithm suitable for environments with discrete action spaces and complex dynamics. PPO is chosen for its stability and sample efficiency, making it well-suited to the task of dynamically managing hedge positions for a Uniswap v3 LP position. The objective

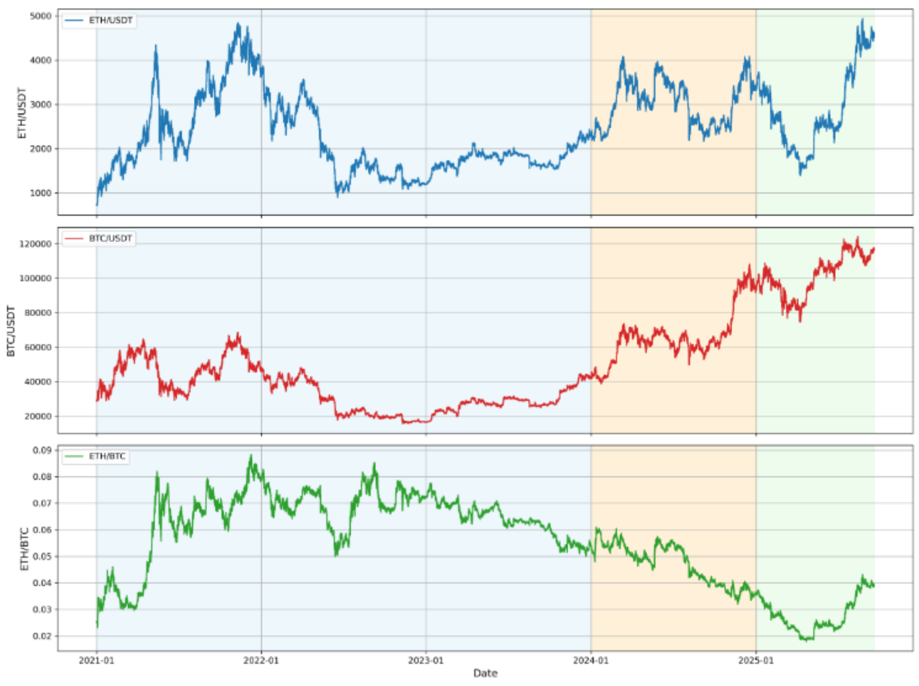

Figure 6: Historical price series for ETH/USDT, BTC/USDT, and the ETH/BTC ratio. The shaded regions indicate the training (blue), validation (orange), and testing (green) splits.

of the training process is to learn a policy that reduces downside risk, while retaining some exposure in trending markets, balancing risk mitigation with potential upside participation.

Training episodes are constructed from historical price data sampled at 15-minute intervals, capturing realistic short-term market dynamics of the underlying tokens. To improve robustness and prevent overfitting to specific market scenarios, the starting point of each episode is randomized within the available dataset, exposing the agent to a diverse range of market conditions, including periods of high volatility and relatively stable prices. Each episode proceeds for maximum of 9000 15-minute timesteps (approx. 94 days), corresponding to a meaningful horizon for LP evolution. The environment simulates the passive evolution of the LP within its specified range, while allowing the agent to actively manage the hedge positions at each timestep. Episodes terminate either upon reaching the maximum number of steps or if the LP position moves outside its predetermined price range, ensuring consistent and controlled learning experiences.

The PPO agents were trained using the Stable-Baselines3 implementation with the default hyperparameters, presented in Table 4. The algorithm uses an MLP policy (`MlpPolicy`), a feedforward neural network with fully connected layers, to map the 20-dimensional state of the Uniswap LP environment to discrete hedge actions. The network employs `tanh` activation functions for all layers. The network architecture is shown in Table 5.

Table 4: Hyperparameters used to configure PPO.

Name	**Value**
learning_rate	0.0003
batch_size	64
n_steps	2048
n_epochs	10
gae_lambda	0.95
ent_coef	0.0
gamma	0.99
n_envs	8
total_timesteps	1,000,000

Table 5: Network architecture hyperparameters used for the PPO agents.

Name	**Value**
Feature extractor layer size	64
Actor hidden layers	64, 64
Critic hidden layers	64, 64
Activation function	tanh

To stabilize training and expose the agent to diverse market conditions, eight parallel environments were employed. Each environment starts at a random point in the training data, allowing the agent to collect experience from different trajectories simultaneously. Episodes are reset automatically once they reach the maximum length or the LP position moves outside its price range. Training was performed for roughly 1,000,000 timesteps per run, with snapshots saved at 100,000-step intervals (i.e., 100,000, 200,000, 300,000, etc.) to monitor learning progress. Since the training did not stop exactly at 1,000,000 timesteps, a final snapshot was also saved at the actual final timestep (e.g., 1,015,808), resulting in 11 snapshots per run. Variable normalization and scaling were applied as described in Tables 1, 2, and 3.

For agent selection, each training run was evaluated at these snapshots. With 10 independent runs using different random seeds, this results in a total of 110 candidate agents (10 runs × 11 snapshots per run).

Each candidate agent was evaluated on the validation set using fixed-length episodes starting at multiple random points to ensure coverage of diverse market conditions. For each episode, the agent interacts with the Uniswap LP environment using the learned policy without further training, and the primary performance metric used is the Sortino ratio, which emphasizes downside risk relative to expected returns. It is defined as

$$\text{Sortino Ratio} = \frac{\bar{R}_t - R_f}{\sigma_d}, \tag{13}$$

where $\bar{R}_t$ is the mean one-step portfolio return within the episode, R_f is the risk-free rate (set to 0 in this thesis), and σ_d is the downside deviation of timestep returns, calculated as

$$\sigma_d = \sqrt{\frac{1}{T}\sum_{t=1}^{T} \min(0, R_t - R_f)^2}, \tag{14}$$

with R_t representing the portfolio return at timestep t and T the total number of timesteps in the episode. Only returns below the target threshold are included in σ_d, so upside volatility does not penalize the metric.

The performance of each candidate agent is then averaged across multiple episodes and across all validation start points. The snapshot achieving the highest average Sortino ratio is identified for every training run. From the 10 independent runs (with different random seeds), the snapshot with the best policy is selected per run, resulting in 10 candidate agents in total. These "best-per-seed" agents are then used for further testing and comparison. The reason the Sortino ratio was chosen as the validation criterion is that the agent is trained to maximize portfolio returns without explicit penalties for volatility or drawdowns. Using the Sortino ratio as a selection metric filters out agents that may achieve high returns but also exhibit large downside volatility, thereby aligning agent evaluation with the intended training objective. This procedure ensures that agent selection is robust to stochastic variations in training and reflects consistent performance across the validation dataset, rather than being dependent on a single random initialization or snapshot.

5.3 Baseline Hedging Strategies

To evaluate the performance of the reinforcement learning agent, we compare it against a set of baseline hedging strategies that represent common or intuitive approaches to risk management. These baselines serve as reference points, ranging from no hedging at all to simple rule-based dynamic hedging schemes. Each baseline is implemented consistently with the same environment dynamics and transaction cost assumptions used for the PPO agent.

No hedge. The most basic reference strategy is to leave the LP position completely unhedged. This reflects the pure exposure of the LP position to market movements. While it maximizes participation in upside trends, it also leaves the portfolio fully vulnerable to downside volatility. This baseline provides a lower bound on risk management effectiveness. When analyzing singular LP positions, depending on the market conditions no hedge is most likely either the best or the worst strategy.

Passive hedge. At the beginning of each episode, a hedge is placed to offset a portion of the LP's total portfolio value. For example, if the LP position consists of equal value in BTC and ETH, a *full hedge* offsets 100% of the portfolio by taking opposing positions equal to the full starting exposure in both BTC and ETH. A *half hedge* offsets 50% of the total portfolio value, implemented by hedging half of the starting BTC exposure and half of the starting ETH exposure. Once placed, the hedge remains fixed throughout the episode without further rebalancing. Passive hedging is straightforward to implement, requiring no active management after initiation, which makes it a practical strategy for retail traders. It therefore serves as a natural benchmark for evaluating the PPO agents.

Fixed-frequency rebalancing. In this strategy, the hedge is adjusted back to neutrality at predetermined time intervals, regardless of market conditions. This approach reflects practical rules that traders may follow due to operational constraints. Fixed-frequency rebalancing is straightforward to implement, even for retail traders, as adjustments are made only at specific time points. Its main limitation is that it may rebalance too frequently during calm market periods (incurring unnecessary transaction costs) or too infrequently during volatile periods (allowing risk to accumulate). Another weakness is that this strategy always incurs losses when fee revenue is neglected. Since this strategy rebalances to a fully hedged at a fixed frequency, every rebalancing incurs LVR thus leading to certain losses without the fee revenue. In this thesis, we test multiple frequencies, including daily, bi-weekly, and weekly rebalancing, and since the strategy cannot generate positive returns, report the results for the frequency that produced the shallowest average drawdowns on the test set as a baseline.

Fixed-deviation rebalancing. In this strategy, the hedge is adjusted only when the LP composition deviates beyond a predefined threshold from the composition at the last rebalance. This event-driven approach monitors the relative token weights in the LP, rather than relying on fixed time intervals, and triggers rebalancing only when meaningful changes occur. By doing so, it avoids unnecessary transaction costs from small fluctuations while remaining responsive to significant shifts in the LP position. In this thesis, thresholds from 1% to 10% in 1% increments are tested. As with the fixed-frequency strategy, only the results corresponding to the threshold that produced the smallest average drawdowns on the test set are reported in the main results section.

The four baseline strategies described above are summarized in Table 6, which provides a concise overview of each approach. By comparing the PPO agent to these baselines, we can evaluate how much additional value the learned policy provides in terms of risk-adjusted performance and responsiveness to market dynamics.

Table 6: Baseline hedging strategies used for comparison with the PPO agent.

Strategy	**Description**
No hedge	LP remains fully exposed, with no hedging performed.
Passive hedge	Initial hedge placed to neutralize LP exposure at $t = 0$, held fixed throughout.
Fixed-frequency rebalancing	Hedge rebalanced to neutrality at predetermined fixed intervals.
Fixed-deviation rebalancing	Hedge rebalanced when price deviation exceeds predetermined % threshold.

5.4 Model Assumptions and Limitations

To ensure clarity and reproducibility, several simplifying assumptions are made in the environment and baseline implementations:

- **Price data approximation:** Binance spot price data are used as a proxy for Uniswap v3 LP prices. Following [34], arbitrage activity ensures that the LP price deviates at most by the pool fee (0.3% for the ETH/BTC pool) from the global market price, justifying the use of centralized exchange data.
- **Funding rates:** Funding rates are assumed constant over each 15-minute interval and interpolated where necessary.
- **Transaction costs:** All hedge adjustments, for both the PPO agent and baseline strategies, incur proportional transaction costs, modeled as 0.3% of the traded notional per adjustment. These costs, which include slippage and the Binance trading fee, are deducted immediately from the portfolio value.
- **Price-taking and infinite depth assumption:** Futures markets are assumed to have infinite depth, and the agent's trades do not impact market prices or other participants.
- **Discrete decision intervals:** Hedging decisions occur at the discrete timesteps defined by the environment (15-minute intervals), rather than continuously.
- **Baseline consistency:** All baseline strategies and the PPO agent interact under the same environment rules, including price data, LP evolution, and cost model.

Additionally, fee revenue generated by the LP position is not included in the simulation, focusing the analysis solely on hedging and market exposure. These assumptions are intended to approximate real trading conditions while maintaining computational tractability and should be considered when interpreting results and comparing strategies.

6 Results

6.1 Validation Results

All candidate agents were evaluated on the 2024 validation dataset, using 10 fixed starting points shared across all seeds to ensure comparability. For each seed, the best snapshot was selected according to the highest mean Sortino ratio. Table 7 summarizes the performance of these best-per-seed agents in terms of the mean Sortino ratio, mean episode return, and maximum drawdown.

Table 7: Validation results for the best snapshot per seed. Columns show the training seed, selected training timesteps at the snapshot, Sortino ratio of one-step portfolio returns per episode, mean episode portfolio return, and maximum drawdown observed during validation episodes. At the bottom mean values of the Sortino ratio and episode returns, as well as worst drawdown in the validation is presented.

Seed	Snapshot	Sortino	Return (%)	Max Drawdown (%)
1	900K	-0.0056	0.31	4.35
2	400K	-0.0026	-1.31	8.85
3	200K	-0.0065	-0.40	14.79
4	600K	0.0060	-0.47	11.35
5	400K	0.0003	0.30	6.42
6	300K	-0.0060	-0.12	11.58
7	400K	-0.0026	-0.90	8.05
8	700K	0.0000	-0.18	8.02
9	1.02M	-0.0034	-0.73	10.09
10	500K	-0.0034	0.09	9.04
Combined	–	**-0.0025**	**-0.34**	**14.79**

The results show that the best-performing snapshot varies considerably across training seeds. While some agents achieved their highest validation performance relatively early in training (e.g., at 200K or 300K timesteps), others continued to improve up to 700K or 900K timesteps, with one agent reaching their best performance at the final snapshot (at 1.02M timesteps). However, it is worth noting that the differences between snapshots within a single seed were generally small. In contrast, the variability in performance across different seeds was larger, which is most visible in Table 7 in Max Drawdown column, indicating that random initialization and stochastic training dynamics have a stronger influence on final agent performance than the choice of snapshot within a single run. This emphasizes the importance of evaluating multiple independent runs rather than relying on a single training instance. The mean episode length was 6488 steps.

The Sortino ratio, which emphasizes downside risk relative to expected returns, provides a measure of how well each agent manages negative portfolio fluctuations. Since the ratio is computed from 15-minute returns, its absolute values are naturally small. Across the best snapshots for each seed, the Sortino ratios are generally near zero, with two out of ten agents achieving positive values. By averaging performance across multiple episodes and computing the Sortino ratio from one-step returns, only about half of the selected snapshots corresponded to the highest mean return or the smallest maximum drawdown. Still, this is to be expected since the aim of the validation procedure was to select agent snapshots that exhibit the least tendency to experience quick large drawdowns, aligning snapshot selection with the risk management objective of the PPO agent.

Even though the mean returns were similar, maximum drawdowns differed significantly between seeds, and some snapshots outperformed others by a wide margin, all ten best-per-seed agents were selected for the testing phase.

6.2 Evaluation Metrics

Each of the agents and baselines are tested over the same set of episodes to ensure a fair comparison. The testing procedure selects 10 random starting points from the previously defined test set. For each starting point, the LP position and hedge are initialized, and the performance of each strategy is tracked throughout the episode. To simulate the results for a single LP position, each testing run is terminated in the same manner as the training episodes.

Metrics are calculated individually for each episode and then averaged across all episodes to obtain stable estimates of performance. This approach captures both the typical behavior of a strategy and its variability across different market scenarios. The procedure is designed to provide a reasonable expectation of the typical results for a single LP position.

All hedging strategies (PPO agents and the baselines) are evaluated in the same environment, using identical historical data and settings for the LP position and futures. This ensures that differences in performance are attributable to the strategy itself rather than variations in market conditions. To facilitate comparability, metrics are preprocessed where necessary, for example, returns are expressed as percentages relative to the LP position, so that all strategies can be assessed on the same scale.

Selecting appropriate performance metrics is crucial for a fair and meaningful comparison between the PPO hedging strategy and the baselines. In this thesis, the primary goal is to evaluate how effectively each strategy hedges a single LP position, rather than to assess absolute profitability. Because all the fully hedged strategies, such as fixed-frequency and fixed-threshold baselines, are inherently negative strategies when fee revenue is neglected, traditional risk-adjusted performance metrics such as the Sortino ratio may be misleading.

All performance metrics are calculated using the total portfolio value at each step, denoted by V_t. This value includes both the LP position and any hedge positions, with transaction costs already accounted for. Using portfolio values allows a direct comparison between strategies by evaluating how effectively each strategy hedges the LP position.

Given this, we define three complementary evaluation metrics:

1. **Maximum Drawdown (MaxDD)** Measures the worst-case loss of the portfolio within an episode. For an episode of length T, it is defined as

$$\mathrm{MaxDD} = \max_{t\in[1,T]} \frac{V_{\mathrm{peak},t} - V_t}{V_{\mathrm{peak},t}}\,, \tag{15}$$

   where V_t is the portfolio value at step t and $V_{\mathrm{peak},t} = \max(V_0, V_1, \ldots, V_t)$ is the highest portfolio value reached up to step t.

   When multiple episodes are evaluated, the **mean maximum drawdown** is computed as

$$\overline{\mathrm{MaxDD}} = \frac{1}{N_{\mathrm{Ep}}} \sum_{i=1}^{N_{\mathrm{Ep}}} \mathrm{MaxDD}^{(i)}\,, \tag{16}$$

   where $\mathrm{MaxDD}^{(i)}$ denotes the maximum drawdown observed in episode i.

2. **Mean Episode Return** Expresses the mean portfolio return over the episode relative to the initial portfolio value

$$R_{\mathrm{Ep}} = \frac{V_T}{V_0} - 1\,, \tag{17}$$

   where V_0 is the initial portfolio value and V_T is the portfolio value at the end of the episode. When multiple episodes are tested, the **mean episode return** is

$$\bar{R}_{\mathrm{Ep}} = \frac{1}{N_{\mathrm{Ep}}} \sum_{i=1}^{N_{\mathrm{Ep}}} R_{\mathrm{Ep}}^{(i)}\,, \tag{18}$$

   where N_{Ep} is the number of episodes evaluated.

3. **Standard Deviation of Episode Returns (Ret STD)** Measures the variability of total portfolio returns across episodes

$$\sigma_{Ep} = \sqrt{\frac{1}{N_{\mathrm{Ep}} - 1} \sum_{i=1}^{N_{\mathrm{Ep}}} \left(R_{\mathrm{Ep}}^{(i)} - \bar{R}_{\mathrm{Ep}}\right)^2}\,. \tag{19}$$

These metrics together provide a comprehensive view of both risk and relative performance, allowing for an interpretable comparison between the adaptive PPO strategy and the static baselines.

6.3 Results

The evaluation results for all baseline strategies and PPO agents are summarized in Table 8. Each value represents the mean across ten evaluation episodes, with returns and drawdowns expressed as percentages relative to the initial portfolio value. The standard deviation of episode returns captures variability in performance across episodes, while the worst maximum drawdown highlights each strategy's most severe observed loss. The mean episode length for the testing set was 4506 steps, which is on around 2000 steps less on average compared to the validation set.

The results confirm that unhedged exposure (`NoHedge`) exhibits the highest variability, with a mean episode return of approximately 21.9% but a large mean maximum drawdown of 14.4% and a worst-case drawdown exceeding 33%. This reflects the large price swings inherent in the underlying LP position when no hedging is applied and also shows that on average the testing set had increasing ETH and BTC prices. Conversely, the fully hedged baselines (`PassiveHedge100`, `FixedFreq48` and `Deviation10`) exhibit much lower drawdowns and return variability but are consistently negative in expectation, as transaction costs and LVR dominate in the absence of fee revenue. Since the `FixedFreq48` and `Deviation10` baselines outperform the fully hedged `PassiveHedge100` strategy, the subsequent analysis focuses on comparing the PPO agents against these stronger baseline benchmarks. In addition it is worth noting that the `FixedFreq48` and `Deviation10` produced very similar results in all testing categories and were consistent between episodes.

As expected, in a rising market the partially hedged baseline (`PassiveHedge50`) achieves better returns on average than the more hedged counterparts, but compared to `FixedFreq48` and `Deviation10` the standard deviations of returns and drawdowns are considerably larger.

Across the PPO agents, performance varied depending on the random seed. The agentss with the smallest maximum drawdowns and return standard deviations (`Seed1` and `Seed9`) also produced the most negative mean returns, showing results very similar to the static baselines `FixedFreq48` and `Deviation10`. Interestingly, `Seed1` was also the best-performing agent on the validation set, achieving both the highest mean return and the lowest drawdown. The `Seed3` agent achieved the highest mean return but also exhibited the largest return standard deviation and mean drawdown, indicating higher volatility and risk. In contrast, the `Seed6` agent, which also produced positive average episode returns, showed variability and drawdown levels close to the overall mean across all PPO agents.

Overall, five out of the ten PPO agents produced mean returns between -2% and 0%, and seven agents exhibited maximum drawdowns between 7% and 10%, indicating that the agents generally performed within a narrow and consistent range. Three agents experienced losses exceeding 13% at some point during testing, which is notably higher than the maximum drawdown of the `Deviation10` baseline (7.73%). It is important to note, however, that the baselines represent optimized strategies designed for this environment, while the PPO agents correspond to those that performed best during validation. When comparing the combined PPO results to the strongest baselines (`FixedFreq48` and `Deviation10`), the PPO agents achieved a less negative mean episode return (-0.91%) than both baselines (-4.51% and -3.86%, respectively). Nevertheless, the PPO agents also displayed greater variability, with approximately twice the return standard deviation of the baselines. In contrast, the mean maximum drawdowns were broadly similar across the PPO and baseline strategies, suggesting that the reinforcement learning agents were able to maintain comparable downside protection while achieving slightly higher average returns.

Table 8: Summary of the testing set performance from the 10 testing episodes. Columns show the name of the baseline or PPO distinguished by the training seed, mean episode return, standard deviation of episode returns, mean maximum drawdown, and worst maximum drawdown observed during episodes. PPOcombined refers to combined results from all tested PPO agents.

Name	$\bar{R}_{\text{Ep}}$ (%)	σ_{Ep} (%)	$\overline{\text{MaxDD}}$ (%)	MaxDD (%)
Deviation10	-3.86	1.85	3.87	7.73
FixedFreq48	-4.51	2.11	4.54	9.21
NoHedge	21.91	19.74	14.44	33.43
PassiveHedge100	-8.99	6.18	10.33	18.58
PassiveHedge50	6.44	9.41	8.05	19.03
PPO_Seed1	-4.38	1.46	4.62	7.00
PPO_Seed2	-1.39	5.09	4.86	14.81
PPO_Seed3	3.30	5.92	5.97	13.35
PPO_Seed4	-1.55	3.63	4.60	9.52
PPO_Seed5	-0.30	3.62	3.95	9.05
PPO_Seed6	1.83	4.55	4.46	9.62
PPO_Seed7	-2.30	4.89	5.92	19.53
PPO_Seed8	-0.80	4.63	4.76	9.90
PPO_Seed9	-3.21	1.46	4.00	7.00
PPO_Seed10	-0.25	2.98	3.69	8.07
PPO_Combined	-0.91	4.45	4.68	19.53

6.4 PnL Trajectories and Model Performance

To further compare the performance of the agents, Figures 7 and 8 illustrate the normalized PnL trajectories across selected episodes. In each figure, the upper subplot presents the percentage changes of `ETH/USDT`, `BTC/USDT`, and the high-variance baselines (`NoHedge` and `PassiveHedge50`), providing an overview of the underlying LP asset trends. The lower subplot displays the lower-variance baselines (`FixedFreq48`, `Deviation10` and `PassiveHedge100`) together with the best-, median-, and worst-performing PPO agents for that episode.

During Episode 4, the price of `ETH/USDT` increased by approximately 80%, while `BTC/USDT` rose by around 30%. After roughly 1000 steps, even the lower bound of the PPO agents one-standard-deviation band surpasses the returns of all baseline strategies. The baselines `FixedFreq48` and `Deviation10` exhibit almost identical behavior, characterized by very low variance between steps and a stable, gradual rate of change throughout the episode. In contrast, the PPO agents display more dynamic behavior, with multiple shifts of several percentage points during the same period, reflecting their adaptive nature.

Figure 8 illustrates an episode where both `ETH/USDT` and `BTC/USDT` perform poorly. Around step 2500, the `ETH/USDT` price declines by over 20% within approximately 500 steps, while `BTC/USDT` drops by about 10%. The baselines `FixedFreq48` and `Deviation10` again display the same stability observed in Figure 7, with their returns trending downward at a steady rate, except for small dips around step 250 and near the end of the episode—periods when the ETH and BTC prices briefly diverge. However, this variation remains minor compared to the volatility observed in the PPO agents.

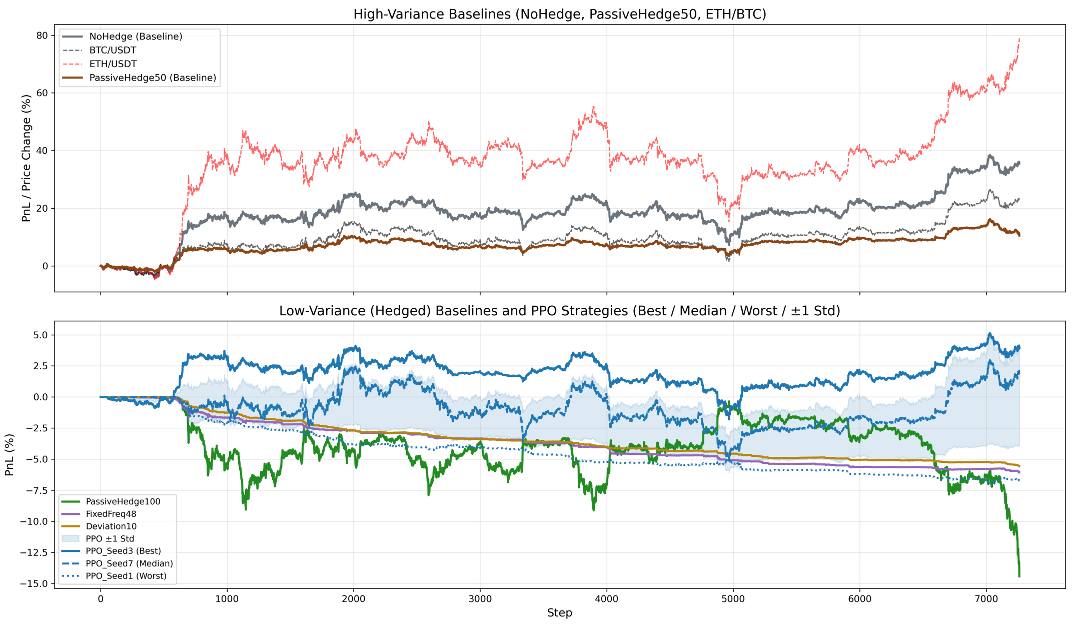

Figure 7: Normalized PnL trajectories for baseline hedging strategies and PPO agents during Episode 4. The upper panel shows high-variance baselines (`NoHedge`, `PassiveHedge50`) and normalized ETH/BTC price paths. The lower panel compares low-variance baselines with PPO strategies, where the blue shaded area represents the mean performance of all PPO agents ±1 standard deviation. The solid, dashed, and dotted blue lines correspond to the best, median, and worst PPO agents, respectively.

The PPO agents appear to struggle when both `ETH/USDT` and `BTC/USDT` experience simultaneous declines. Nonetheless, even when the worst-performing PPO agent reaches approximately -6% near step 2800, the median agent remains above the `FixedFreq48` and `Deviation10` baselines. Only the sharp market downturn at the end of the episode causes the PPO agents to underperform relative to the baselines. Despite this, the median PPO agent ends with a loss of around -7.5%, while the unhedged `NoHedge` strategy experiences a significantly larger drawdown of roughly -25%. This demonstrates that, although the PPO strategies incur losses under adverse conditions, the hedging decisions still mitigate a substantial portion of downside risk.

Examining the first 2500 steps, the ± 1 standard deviation band of PPO performance narrows from roughly $[-2\%, 0\%]$ during the initial 1800 steps to $[-4\%, 0\%]$ by step 2500. During the same interval, `ETH/USDT` experiences a 20% drop (steps 50–300), a 15% rally, and another 15% decline—price movements that closely mirror the unhedged `NoHedge` trajectory. These patterns suggest that, despite the volatility, the PPO agents are capable of adapting their hedge positions to offset large price swings and provide partial downside protection.

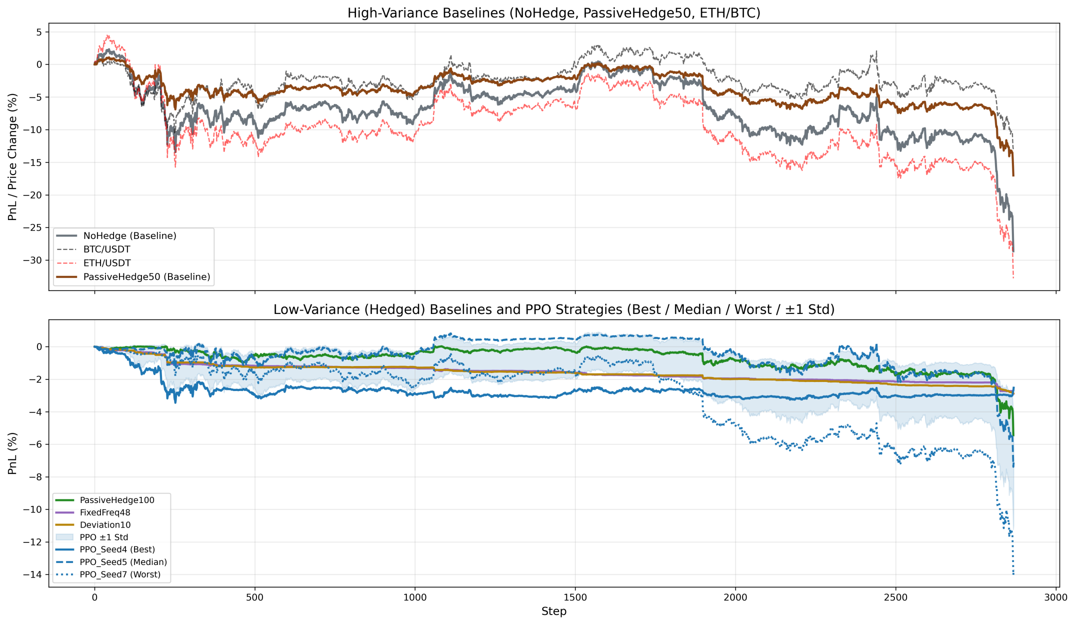

Figure 8: Normalized PnL trajectories for baseline hedging strategies and PPO agents during Episode 7.The plot details follow the same conventions as in Figure 7.

6.5 Hedge Ratio Actions and Model Behavior

To complement the PnL trajectory analysis, Figures 9 and 10 present the *chosen hedge ratio actions* taken by the PPO agents during Episodes 4 and 7. These plots illustrate how each agent's target hedge ratios for ETH and BTC evolve throughout the episode, together with the mean behavior across all trained agents. Only the same three individual PPO agents shown in the PnL trajectory figures 7 and 8 are included here to keep the plots readable. The shaded regions represent the ± 1 standard deviation range of hedge ratios across all ten trained agents, providing a measure of consensus or divergence between agents. Including the underlying asset price dynamics in the same plots allows for a direct visual comparison between market movements and the timing of hedge adjustments. For clarity, the individual hedge ratio trajectories are lightly smoothed using a short moving average to reduce visual noise while preserving their overall dynamics.

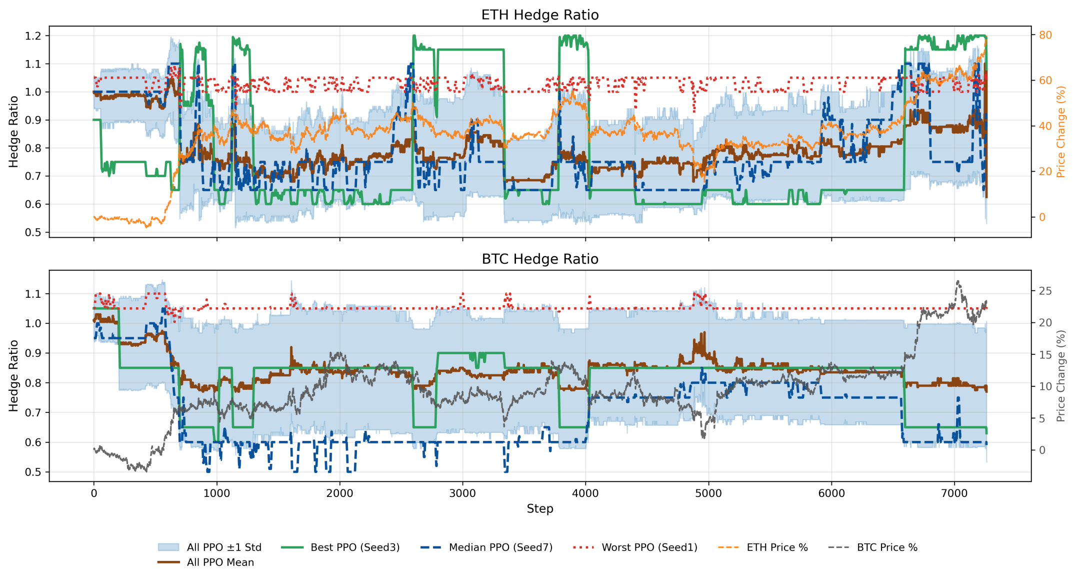

Figure 9: Chosen hedge ratio actions for PPO agents and the underlying (ETHUSDT or BTCUSDT) price change during Episode 4. The top panel shows the evolution of the ETH hedge ratio actions and the bottom panel shows the BTC hedge ratio actions taken by the PPO agents. The solid brown line represents the mean action across all PPO agents, with the shaded region indicating ±1 standard deviation. The green, blue, and red lines correspond to the best-, median-, and worst-performing PPO agents, respectively, based on final portfolio value in this episode. The figure illustrates how the agents' hedge ratio decisions evolve over time and differ across agents.

Across both episodes, the PPO agents exhibit markedly different hedging behaviors, as evidenced by the variation in individual actions and the wide standard deviation bands. This indicates that the agents' decisions contain a substantial degree of stochasticity and that their behaviors do not converge toward a single stable policy. Furthermore, the results suggest that the agents' actions are highly sensitive to the particular training process and initialization, which is an expected outcome when learning from normalized financial time series data.

The mean action trajectories tend to evolve more smoothly than those of individual agents, which often display stepwise changes reflecting discrete policy decisions. Periods of higher variability within the shaded region correspond to moments when the agents disagree on hedge direction or magnitude, often coinciding with increased market volatility or rapid price reversals.

During Episode 4, which features a sustained rise in both ETH and BTC prices, the PPO agents reduce their target hedge ratios from approximately 1.0 to 0.8 between steps 800 and 1000, indicating that the agents, on average, anticipated the upward trend to continue. The average hedge ratio remains within [0.7, 0.9] for ETH and [0.8, 0.9] for BTC, with most fluctuations in the average target hedge ratios closely following volatile movements in the underlying prices. For instance, around step 3500, the ETH target hedge ratio declines from 0.8 to 0.7 coinciding with a sharp drop in the ETH price. Overall, during this uptrend, the agents' average hedging behavior appears sensible: the target hedge ratio tends to increase slightly when prices rise and decrease when prices fall.

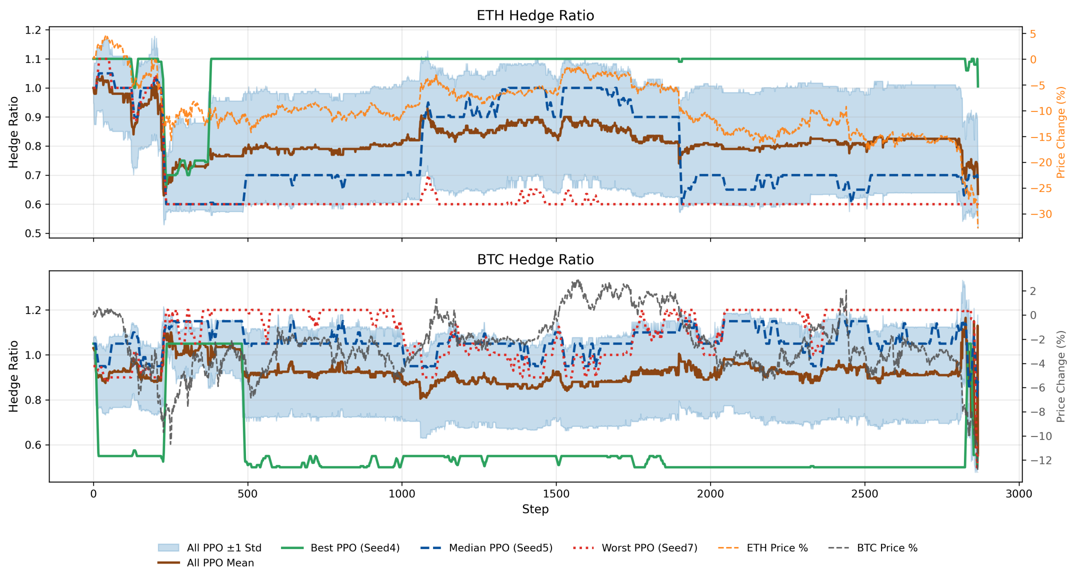

Figure 10: Chosen hedge ratio actions for PPO agents and underlying prices during Episode 7. The figure shows the evolution of hedge ratios for ETH and BTC. Compared to Episode 4, the agents exhibit more volatile and divergent hedge adjustments as prices decline, reflecting differing risk responses among the agents.

In contrast, Episode 7 exhibits falling asset prices and a more volatile policy response. At the beginning, the average hedge ratio for ETH almost perfectly times the bottom of a short-term downward movement around step 250. Thereafter, the hedge ratio evolves reasonably until approximately step 1500, when a more pronounced downtrend in ETH price begins. The average hedge ratio continues to decrease gradually as prices fall, reaching its lowest point toward the end of the episode, coinciding with a sharp final drop in ETH price. For BTC, the average target hedge ratio is considerably more stable, remaining within [0.8, 1.0] for most of the episode. Between steps 1500 and the end, BTC prices decline by roughly 15%, which the agents fail to anticipate, maintaining an almost unchanged hedge ratio throughout the downturn.

Overall, these observations demonstrate that the PPO agents adapt their hedge ratios in a directionally consistent manner with underlying price movements decreasing the target hedge ratio when prices decline and increasing it when prices rise thereby reducing exposure during upward movements and increasing exposure during downward movements. The behavior was broadly similar across both reviewed episodes. The agents’ main shortcoming appears to be reducing the hedge too quickly during downtrends, which can lead to significant drawdowns, as illustrated in Figure 8. This highlights both the agents’ ability to learn meaningful hedging behavior and the remaining limitations in their stability and robustness across different market conditions.

7 Discussion and Conclusions

This thesis investigated how effectively a Deep Reinforcement Learning agent learns to manage the hedge of a Uniswap v3 liquidity provider position using cryptocurrency futures. The objective was to minimize downside variance, while keeping exposure to upward price movements. To address this objective, a custom simulation environment was developed to replicate the value dynamics of a Uniswap v3 LP position alongside a corresponding futures hedge. The environment models the LP's exposure to price movements, liquidity range, the PnL of the hedge position and transaction costs from rebalancing. Ten Proximal Policy Optimization agents were trained within this environment to dynamically adjust the hedge ratio based on observed market conditions and volatility signals derived from historical cryptocurrency data. The agents were then evaluated against several baseline hedging strategies, including static and rule-based approaches, by comparing overall drawdowns, episode returns and their variance.

The results across the test set showed substantial variation between agents, with mean episode returns ranging from –4.38% to 3.30% and maximum drawdowns from 19.53% to 7.00%. Despite this variability, the worst-performing agents achieved mean episode returns comparable to those of the best-performing baseline strategies, while the average return across all PPO agents exceeded that of the baselines. The PPO agents exhibited mean drawdowns similar to the baselines in most episodes, although three out of ten agents experienced maximum drawdowns that were at least twice as large as those of the baselines. Overall, these findings suggest that the PPO agents learned moderately profitable policies, as they maintained comparable drawdowns to the baselines while achieving generally higher (or less negative) average returns.

Although the results generally indicate that the PPO agents learned to hedge the LP positions moderately well, several key observations should be noted. The results were more stable in the validation set, where the mean episode length was 6,488 steps compared to 4,506 steps in the testing set. Since an episode terminates before 9,000 steps only when the ETH/BTC ratio price moves outside the underlying liquidity pools range, this suggests that the price dynamics were more stable during the validation period. Consequently, the PPO agents appear to perform better in calmer market environments with fewer abrupt price changes.

Furthermore, the action analysis revealed that the PPO agents often exhibited significantly different views on the appropriate hedge ratio at a given time. This behavioral divergence is likely attributable to the normalization of the environment, the relatively limited size of the training dataset, and the fast training procedure, all of which can contribute to variability and stochasticity in the learned policies.

An important methodological aspect influencing the learning outcomes is the normalization of all state variables in the environment. Each of the 20 features in the observation vector was normalized to ensure numerical stability and reduce non-stationarity between episodes starting at different market levels. While this approach facilitated faster and more stable PPO training, it also removed information about absolute price levels and market regimes. As a result, the agents were forced to learn policies purely based on relative changes rather than absolute magnitudes, making it more difficult to distinguish between different volatility and trend environments. This likely contributed to the divergent behaviors observed across agents trained on the same data, since normalization amplifies the influence of local patterns and short-term dynamics while masking broader market context. The normalization, the relatively small training dataset of approximately 105,000 time steps and short episodes of maximum 9000 steps further intensified this effect, limiting the diversity of market conditions the agents could experience. Keeping the dynamics generalized resulted in different policies between agents, but potentially helped the agents to have relatively stable results in both validation and test sets. The main drawback of the hedging policies seemed to be too aggressive reduction of the hedge when the underlying prices decrease. In future work regime-aware features could help preserve essential market structure information while maintaining the numerical stability required for reinforcement learning.

Beyond the previously discussed limitations, several other aspects of the training environment restrict the generality of the results. First, the simulation framework models only the price risk of the Uniswap v3 LP position and does not account for fee revenue generated from providing liquidity. In real LP positions, fee income offsets part of the downside exposure, effectively altering the optimal hedge ratio. Excluding this component simplifies the optimization problem but may lead to hedge ratios that are overly defensive compared to those used in live market conditions.

Second, the liquidity allocation of the LP position is assumed to remain fixed throughout each episode. In practice, LPs actively rebalance their price ranges to maintain capital efficiency and reduce impermanent loss. The absence of this behavior means that the agent's task was restricted to managing only the external futures hedge, not the joint optimization of liquidity placement and hedge exposure.

Third, the execution of the hedge trades is modeled as frictionless apart from proportional transaction costs and a simplified funding rate input. Real-world trading introduces additional complexities, including slippage and variable liquidity depth. These factors can materially affect both hedge efficiency and cost, and may change the relative attractiveness of dynamic versus static hedging policies.

Finally, the training setup is limited to a single ETH/BTC liquidity pool, and the reported results represent mean outcomes across multiple independent episodes of varying lengths. Each episode captures the agent's performance within a specific price trajectory and liquidity range, but the analysis does not encompass a continuous, long-term sequence of consecutive LP positions. As a result, the evaluation reflects short- to medium-term hedging behavior rather than sustained portfolio management over extended market cycles. This limitation should be considered when interpreting the results, as the performance of both the LP and the hedge may differ when range resets, accumulated fees, or regime transitions are taken into account.

Despite these limitations, the results provide valuable insights into the behavior of the PPO agents and their effectiveness in managing liquidity provider hedges. The agents exhibited behavior that was directionally consistent with underlying market movements. On average, the PPO agents increased exposure during rising markets and reduced exposure during downturns, suggesting that they successfully captured the fundamental relationship between LP value changes and hedge adjustments. This indicates that the reinforcement learning framework was able to internalize a form of adaptive risk management logic. However, the agents occasionally reduced their hedge ratios too aggressively during downward movements in asset prices, leading to larger drawdowns in several episodes. This behavior likely reflects the agents' tendency to react strongly to short-term price shifts within the limited episode horizon, emphasizing the trade-off between responsiveness and stability that is inherent in PPO-based policies.

Overall, the simplifications mean that the trained PPO agents operated in a stylized and controlled environment. The resulting policies therefore represent an early proof of concept of how reinforcement learning can learn dynamic hedging behavior under idealized conditions. Extending the environment to include fee income, dynamic range management, and more realistic market frictions would be essential for assessing the robustness and practical viability of DRL-based hedge management in real-world DeFi markets.

References

[1] Coinmarketcap. https://coinmarketcap.com/charts/. Accessed on 9.4.2025.

[2] defillama. https://defillama.com/dexs/uniswap. Accessed on 9.4.2025.

[3] H. Adams, N. Zinsmeister , S. Moody, R. Keefer, D. Robinson. Uniswap v3 whitepaper. https://app.uniswap.org/whitepaper-v3.pdf, 2021.

[4] S. Loesch, N. Hindman, N. Welch, M. B. Richardson. Impermanent Loss in Uniswap v3. https://arxiv.org/pdf/2111.09192, 2021.

[5] Á. Cartea, F. Drissi, M. Monga. Decentralised Finance and Automated Market Making: Predictable Loss and Optimal Liquidity Provision. https://arxiv.org/pdf/2309.08431, 2023.

[6] L. Heimbach, E. Schertenleb, R. Wattenhofer. Risks and Returns of Uniswap V3 Liquidity Providers. https://arxiv.org/pdf/2205.08904, 2022.

[7] Z. Fan, F. Marmolejo-Cossío, D. Moroz, M. Neuder, R. Rao, D. C. Parkes. Strategic Liquidity Provision in Uniswap v3. https://arxiv.org/pdf/2106.12033, 2021.

[8] H. Zhang, X. Chen, L. F. Yang. Adaptive Liquidity Provision in Uniswap v3 with Deep Reinforcement Learning. https://arxiv.org/pdf/2309.10129, 2023.

[9] J. Du, M. Jin, P. N. Kolm, G. Ritter, Y. Wang, B. Zhang. Deep reinforcement learning for option replication and hedging. *The Journal of Financial Data Science*, 2(4), 44-57, 2020.

[10] H. E. Leland. Option pricing and replication with transactions costs. *The Journal of Finance*, 40(5), 1283-1301, 1985.

[11] R. S. Sutton. Reinforcement Learning: An Introduction, Second edition. The MIT Press, 2018.

[12] P. Yu, J. S. Lee, I. Kulyatin, Z. Shi, S. Dasgupta. Model-based deep reinforcement learning for dynamic portfolio optimization. https://arxiv.org/pdf/1901.08740, 2019.

[13] H. Park, M. K. Sim, D. G. Choi. An intelligent financial portfolio trading strategy using deep Q-learning. *Expert Systems with Applications*, 158, 113573, 2020.

[14] H. Yang, X. Y. Liu, S. Zhong, A. Walid. Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the First ACM International Conference on AI in Finance*, 1-8, October 2020.

[15] J. Cao, J. Chen, J. Hull, Z. Poulos. Deep hedging of derivatives using reinforcement learning. https://arxiv.org/pdf/2103.16409, 2021.

[16] H. Buehler, L. Gonon, J. Teichmann, B. Wood. Deep hedging. *Quantitative Finance*, 19(8), 1271-1291, 2019.

[17] K. Hornik, M. Stinchcombe, H. White. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 359-366, 1989.

[18] I. Goodfellow. Deep learning. The MIT Press, 2016.

[19] R. Bellman. Dynamic programming. Princeton University Press, 1957.

[20] R. S. Sutton, D. McAllester, S. Singh, Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12, 1999.

[21] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 229-256, 1992.

[22] J. Schulman,F. Wolski, P. Dhariwal, A. Radford, O. Klimov. Proximal policy optimization algorithms. https://arxiv.org/pdf/1707.06347, 2017.

[23] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz. Trust region policy optimization. *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, 37, 1889-1897, 2015.

[24] S. Kullback, R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22, 79-86, 1951.

[25] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel. High-dimensional continuous control using generalized advantage estimation. https://arxiv.org/pdf/1506.02438, 2016.

[26] Z. Fan, F. Marmolejo-Cossío, B. Altschuler, H. Sun, X. Wang, D. C. Parkes. Differential liquidity provision in Uniswap v3 and implications for contract design. https://arxiv.org/pdf/2204.00464, 2022.

[27] A. Lipton, V. Lucic, A. Sepp. Unified approach for hedging impermanent loss of liquidity provision. https://arxiv.org/pdf/2407.05146, 2023.

[28] A. Khakhar, X. Chen. Delta hedging liquidity positions on automated market makers. https://arxiv.org/pdf/2208.03318, 2022.

[29] J. Deng, H. Zong, Y. Wang. Static replication of impermanent loss for concentrated liquidity provision in decentralised markets. *Operations Research Letters*, 51(3), 206-211, 2023.

[30] G. Angeris, A. Evans, T. Chitra. Replicating market makers. *Digital Finance*, 5(2), 367-387, 2023.

[31] J. Milionis, C. C. Moallemi, T. Roughgarden, A. L. Zhang. Automated market making and loss-versus-rebalancing. https://arxiv.org/pdf/2208.06046, 2022.

[32] Á. Cartea, F. Drissi, M. Monga. Predictable losses of liquidity provision in constant function markets and concentrated liquidity markets. *Applied Mathematical Finance*, 30(2), 69-93, 2023.

[33] M. Fukasawa, B. Maire, M. Wunsch. Model-free hedging of impermanent loss in geometric mean market makers. https://arxiv.org/pdf/2303.11118, 2023.

[34] G. Angeris, H. T. Kao, R. Chiang, C. Noyes, T. Chitra. An analysis of Uniswap markets. https://arxiv.org/pdf/1911.03380, 2021.

[35] G. Angeris, T. Chitra. Improved price oracles: Constant function market makers. *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 80-91, 2020.

[36] I. Halperin. QLBS: Q-Learner in the Black–Scholes (–Merton) worlds. https://arxiv.org/pdf/1712.04609, 2017.

[37] G. Ritter, P. Kolm. Dynamic replication and hedging: A reinforcement learning approach. *The Journal of Financial Data Science*, 1(1), 159–171, 2019.

[38] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22, 1–8, 2021.