SCHOOL OF
OPERATIONS RESEARCH AND INFORMATION ENGINEERING
COLLEGE OF ENGINEERING
CORNELL FINANCIAL ENGINEERING MANHATTAN
CORNELL UNIVERSITY

Financial Engineering Project

Automated Market Making Strategy in Uniswap V3

Presented to the Faculty of the Graduate School of Cornell University
in partial fulfillment of the
requirements for the Master of Engineering Degree
and the Financial Engineering Concentration

By:
Anton Shchablykin
Shuhan Zhang
Tingxi Ruan
Yu Zhang
Ziang Li

Faculty Advisor:

Sasha Stoikov
December 2022

Faculty Advisor's Signature

Date

Content

Abstract

This paper presents a novel approach to automated market making strategies on the Uniswap V3. The primary objective is to optimize the market making strategy by autonomously adjusting liquidity provisions in response to the changing market conditions. To achieve this, we attempted several simple strategies, a dynamic memory-retaining interval strategy which updated the price bracket by tracking the price movement and developed a reinforcement learning based strategy. Furthermore, we introduce a comprehensive backtesting framework to evaluate the performance of our strategies. This framework simulates market conditions to assess the potential profit and loss under each strategy. Through extensive backtesting, we provide insights into the effectiveness of the market making strategies in managing liquidity and optimizing returns in the dynamic DeFi market. The results demonstrate that the memory

1. Introduction to UniswapV3

1.1 Intro to Uniswap and Automated Market Making

Uniswap stands as a pioneering force in the decentralized finance (DeFi) landscape, introducing a novel approach to asset trading through Automated Market Makers (AMMs). At its core, Uniswap is a decentralized exchange (DEX) built on the Ethereum blockchain, designed to facilitate the seamless exchange of ERC-20 tokens without the need for traditional market makers. Unlike conventional exchanges that rely on order books to match buyers and sellers, Uniswap leverages a unique AMM model. This model employs liquidity pools where tokens are traded against reserves held in smart contracts, rather than against other traders. This innovative mechanism not only democratizes trading by allowing anyone to become a liquidity provider but also ensures constant liquidity and more predictable pricing. The concept of AMMs, popularized by Uniswap, marks a significant departure from traditional finance, heralding a new era of accessibility and transparency in the world of cryptocurrency trading.

1.2 Review of Uniswap V2 and Transactions to V3

Uniswap V2 marked a significant milestone in the decentralized finance (DeFi) landscape, introducing an innovative platform that facilitated automated and permissionless token swaps on the Ethereum blockchain. Central to its operation is the concept of Automated Market Makers (AMMs), a departure from traditional exchange models with order books. In AMMs, liquidity providers (LPs) deposit pairs of tokens into liquidity pools in exchange for trading fees. These pools, instead of relying on traditional market matching, use a constant product formula to determine prices and maintain balance.

The primary revenue model for AMMs like Uniswap V2 involves trading fees. These fees are accumulated with each trade and are distributed to liquidity providers in proportion to their stake

in the pool. This mechanism incentivizes the provision of liquidity, although with inherent risks such as impermanent loss, where the value of deposited tokens diverges from market prices.

However, Uniswap V2 does not give much room for liquidity providers to strategically allocate funds as the protocols will automatically distribute the liquidity evenly across the entire price range. In contrast, Uniswap V3 introduced a more efficient and flexible approach to liquidity provision, and therefore opened more possibilities for LPs to optimize positions. The hallmark of Uniswap V3 is its concentrated liquidity feature, which allows liquidity providers to allocate capital within specific price ranges. This innovation leads to higher capital efficiency, as funds are concentrated where they are most likely to be traded.

Concentrated liquidity, however, comes with new dynamics. It enables higher leverage in liquidity positions, allowing providers to gain greater exposure to price movements within their chosen ranges, potentially augmenting returns. However, this also necessitates more active management of liquidity positions compared to the passive approach prevalent in V2.

Moreover, there is an important cost associated with the freedom to allocate funds: impermanent loss. Impermanent loss is a significant consideration in Uniswap V3. Although liquidity providers can specify a price range much narrower than that in V2 and collect higher fees, they need to face the risk of price going beyond that range. This phenomenon, where the temporary loss is experienced due to the changing price ratio of deposited assets, is more pronounced in V3 because of the focused nature of liquidity positions. Providers need to be strategic in selecting price ranges to mitigate potential losses. They must also consider diversifying their liquidity positions to manage risks effectively.

An often-overlooked aspect of transacting on Uniswap, particularly in its third iteration, is the impact of Ethereum gas fees. The complex and flexible nature of Uniswap V3 can result in higher gas costs, especially when adjusting liquidity positions frequently. Users must account for these fees as they can significantly affect the profitability of trading and liquidity provision on the platform.

In our study, due to the complexity of gas fees calculation, we will assume a constant gas fee for developing our trading strategies.

2. Mechanism of Uniswap V3

2.1 Constant Function Market Makers

The core idea behind the constant function market maker is the product of the reserves must remain unchanged after each trade:

$$x * y = k = L^2 \tag{1}$$

where x and y denote the amount of token x and token y in the pool. when a trader buys token x for token y, the amount of token x increases to $x + \Delta x$, the amount of token y is updated to $y - \Delta y$, and the product of the amounts of two tokens remains k, e.g. $(x + \Delta x) * (y - \Delta y) = k$.

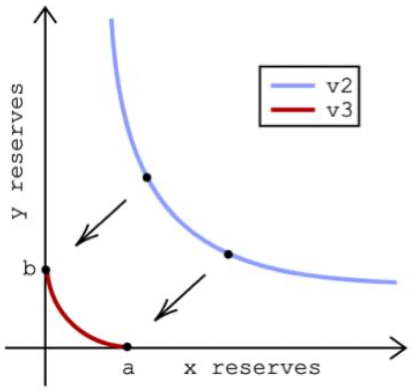

Figure 1. Reserve Function for Uniswap V2 and V3

2.2 Concentrated Liquidity

In Uniswap v2, the liquidity is spread evenly across an infinite price range. This means that the liquidity providers's (LP) funds are available to facilitate trades at any possible price between the pair of assets. However, as most trading activity occurs near the current market price, only a small portion of the liquidity provided by LP is actively being used for trades and LP can only earn fees from the activated liquidity.

Uniswap V3 introduces concentrated liquidity to solve the capital inefficiency in the earlier version, which allows LPs to provide liquidity to a bounded price range, (P_a, P_b). This shifts the reserve function in uniswap v2 to:

$$\left(x + \frac{L}{\sqrt{p_b}}\right)\left(y + L\sqrt{p_a}\right) = k = L^2 \tag{2}$$

By concentrating liquidity in a bounded price range, a higher proportion of the provided capital is active when a swap happens. Typically, a narrower price interval increases the leverage and proportion of transaction fees earned by a liquidity provider (LP). Nonetheless, this narrower interval also amplifies the LP's risk of incurring higher losses if the prices of the assets shift significantly.

2.3 Liquidity Calculation and Assets Amounts

The derivation of liquidity and assets amount can be classified into the following categories: $P \leq P_a, P_a < P < P_b, P_b < P$.

2.3.1 $P \leq P_a$:

From figure 3.1 we can see that if the current price lies outside the price range (P_a, P_b), the liquidity is fully provided by a single asset. Thus, here, the position is fully in x, so set $y = 0$. Substitute to equation (2), we can get:

$$\left(x + \frac{L}{\sqrt{p_b}}\right) L\sqrt{p_a} = L^2 \tag{3}$$

With derivation, we can get the amount of token x:

$$x = L\frac{\sqrt{p_b} - \sqrt{p_a}}{\sqrt{p_b} \cdot \sqrt{p_a}} \tag{4}$$

Correspondingly, the liquidity can be derived as:

$$L = x\frac{\sqrt{p_a} \cdot \sqrt{p_b}}{\sqrt{p_b} - \sqrt{p_a}} \tag{5}$$

2.3.2 $P \geq P_b$:

The derivation is similar to that in the section above.

The amount of token y:

$$\mathrm{y} = \mathrm{L}\left(\sqrt{\mathrm{P_b}} - \sqrt{\mathrm{P_a}}\right) \tag{6}$$

The liquidity is:

$$L = \frac{y}{\sqrt{P_b} - \sqrt{P_a}} \tag{7}$$

2.3.3 $P_a < P < P_b$:

According to a technical note of liquidity math in uniswap V3 (Elsts, 2021), the two assets should contribute to the liquidity equally when they are at the optimal position. In another words, the liquidity provided by asset x in the range (P, P_b) should be equal to that provided by asset y in (P_a, P). Thus, we have:

$$x\frac{\sqrt{p_a}\cdot\sqrt{p_b}}{\sqrt{p_b}-\sqrt{p_a}}=\frac{y}{\sqrt{P_b}-\sqrt{P_a}} \tag{8}$$

And,

$$x=L\frac{\sqrt{p_b}-\sqrt{P}}{\sqrt{p_b}\cdot\sqrt{P}} \tag{9}$$

$$\mathrm{y}=\mathrm{L}\left(\sqrt{\mathrm{P}}-\sqrt{\mathrm{P_a}}\right) \tag{10}$$

Also, we can get liquidity L correspondingly.

2.4 Swap Fees

LPs get incentives for the liquidity they provide. For every trade within the price range, LP get fees proportionally to their share in total pool liquidity.

The calculation of the accrued swap fees is based on the global pool status. Each pool in Uniswap V3 has *feeGrowthGlobal0X128* and *feeGrowthGlobal1X128* to track the total accumulated fees per unit of liquidity. Besides, ticks keep record of fees accumulated below it when the price crosses. Thus, the total fee inside the position is:

$$fee=fee_{global}-f_b(i_l)-f_a(i_u) \tag{11}$$

Where $f_b(i_l)$ denote the fee collected below the lower tick p_b, $f_a(i_u)$ denote the fee collected above the upper tick p_a. The fee earned by the LP is proportional to the total fee in the position due to their amount of liquidity provide to the price interval.

2.5 Impermanent Loss

The term impermanent loss is used to assess the temporary loss of value due to price divergence between the value of the current fee adjusted liquidity position in an AMM, and the HODL value of the position that was originally contributed. To be more specific, the price movement will lead to the change the amounts of tokens for the pair of assets hold by LP, and result in a change in the total value.

Let V_0 denote the total value of the initial holding in terms of asset Y.

$$V_0=y+x*P_0 \tag{12}$$

Substitute x and y with (9) and (10),

We have,

$$V_0 = 2L\sqrt{P} - L\left(\sqrt{P_a} + \frac{P}{\sqrt{P_b}}\right) \tag{13}$$

When swap happens, the price P updated to P', like before, the value of holding kept in the pool is simply substitute P with P', thus,

$$V_1 = 2L\sqrt{P'} - L\left(\sqrt{P_a} + \frac{P'}{\sqrt{P_b}}\right) \tag{14}$$

The value of holding kept outside the pool is:

$$V_{held} = y + x * P' \tag{15}$$

Substitute x and y with (9) and (10),

We have,

$$V_{held} = \mathrm{L}\left(\sqrt{\mathrm{P}} - \sqrt{\mathrm{P_a}}\right) + L\frac{\sqrt{p_b} - \sqrt{P}}{\sqrt{p_b} \cdot \sqrt{P}} * P' \tag{16}$$

To conclude, the impermanent loss is,

$$IL = V_1 - V_{held} \tag{17}$$

2.6 Gas Fee

Gas fees are paid to compensate for the computational energy required to process and validate transaction on the Ethereum network. For LPs, every action taken, e.g. adding or removing liquidity, incurs a gas fee. These fees can vary significantly based on network congestion and the complexity of the transactions. Typically, the gas fee is calculated by the product of cost in computation and gas price. Based on our research, the average gas cost for adding liquidity cost 215k unit and removing liquidity costs 223k unit, and the average gas price during our test period is 22.65 Gwei. Thus, we estimate $8.26 for each action of adding liquidity and $7.27 for each action of removing liquidity as our assumption for gas cost.

3. Data Collection

Our research collected data primarily from two sources: the Etherscan API and the SubGraph QL API. We focused on data of the Uniswap v3 USDC/WETH at 0.05% fee tier as it is where most of the transactions happen.

3.1 Transaction Records with Etherscan API

Using the Etherscan API, we gathered every transaction in the Uniswap v3 framework within a predefined timestamp range. This method returned a list of "event" objects. Then we cleaned and clustered these objects based on the method of the transaction, namely SWAP (exchanging between USDC and WETH with the pool's contract), MINT (creating liquidity to the pool), and BURN (removing liquidity from the pool) etc. These transaction data are key to developing our trading strategies as they provide the most precise price movements compared to other data sources.

	blockNo	timestamp	gasPrice	gasUsed	gasTotal	method	sender
0	17817626	1690862411	18393870038	195792	0.003601	SWAP	0xE592427A0AEce92De3Edee1F18E0157C05861564
1	17817626	1690862411	18393870038	195792	0.003601	SWAP	0xE592427A0AEce92De3Edee1F18E0157C05861564
2	17817626	1690862411	16393870038	178565	0.002927	SWAP	0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD
3	17817627	1690862423	18133456829	623342	0.011303	SWAP	0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD
4	17817627	1690862423	18133456829	212220	0.003848	BURN	NaN
...	...	...	...	...	...	...	...
456731	18560746	1699851419	28981216230	539122	0.015624	SWAP	0xE37e799D5077682FA0a244D46E5649F71457BD09
456732	18560749	1699851455	30561285823	262909	0.008035	SWAP	0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD
456733	18560750	1699851467	52489495067	126056	0.006617	SWAP	0x51C72848c68a965f66FA7a88855F9f7784502a7F
456734	18560753	1699851503	48369775138	126070	0.006098	SWAP	0x51C72848c68a965f66FA7a88855F9f7784502a7F
456735	18560760	1699851587	31581392491	314954	0.009947	SWAP	0xDef1C0ded9bec7F1a1670819833240f027b25EfF

Figure 2. Transaction Data Sample from Etherscan

Figure 3 is a line chart depicting the price movement of Ethereum from August 1, 2023, to November 30, 2023. The x-axis marks the timeline, while the y-axis indicates the price, providing a comprehensive view of Ethereum's price fluctuations over a three-month period with 105 daily observations. This chart is crucial for evaluating the effectiveness of trading strategies against actual market performance. The depicted time frame will be employed for backtesting our trading strategies, thereby enabling a practical assessment of strategy performance in real-world conditions. The chart's data points serve as critical markers for gauging the impact of market events on Ethereum's price.

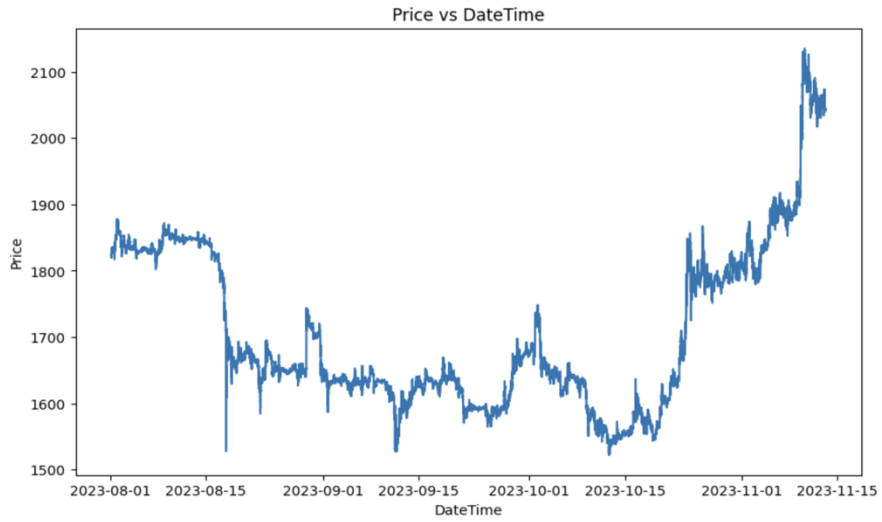

Figure 3. Price Movement from Transaction Data

3.2 Pool's State Data with SubGraph QL API

In addition to Etherscan, we used the SubGraph QL API to supplement our dataset. This API was particularly useful in providing additional information about the blockchain states, which were not as easily accessible through Etherscan. The primary usage of SubGraph is the backtestor we developed, which automatically pulled pool states data and applied any strategy we would like to investigate.

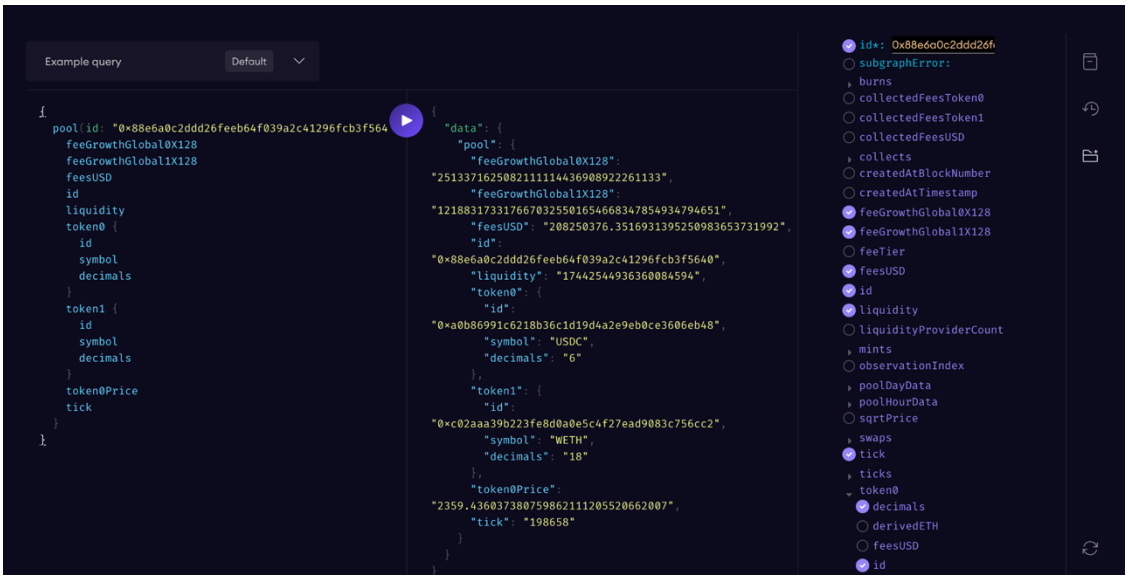

Figure 3. Query Sample from SubGraph

4. Backtesting Framework

4.1 Overview of the Coding Framework in Python

The following sections present a comprehensive coding framework, intricately designed to navigate the complex landscape of Uniswap V3. This framework encompasses various critical aspects: data manipulation, mathematical implementation of Uniswap V3, backtesting mechanisms, and strategy formulation.

Components of the Framework includes:

- Data Manipulation:
    - This segment focuses on extracting and processing data from Uniswap V3. Utilizing advanced querying and data transformation techniques, it lays the groundwork for subsequent analysis.
- Mathematical Implementation in Uniswap V3:
    - A deep dive into the mathematical models underlying Uniswap V3, particularly the liquidity and fee calculations, essential for understanding the dynamics of AMMs.
- **Backtesting Framework**:
    - Here, we explore the development of a backtesting system, crucial for assessing the viability of trading strategies.
- **Strategy Formulation**:
    - This part outlines the methodology for creating and implementing trading strategies, emphasizing the formatting of strategies in coding.

4.2 Framework for Data Manipulation and Analysis (data.py)

4.2.1 Data Retrieval and Processing

The core of our computational approach hinges on the efficient retrieval and processing of data from the Uniswap V3 subgraph. Utilizing GraphQL, a powerful query language, we developed a Python-based framework to extract pivotal financial metrics such as liquidity, trading volume, and token information. This process is encapsulated in the *graph* function, which dynamically constructs and executes a query to fetch hourly and daily data for a specified liquidity pool.

4.2.2 Data Updating Mechanism

The *update* function was designed to ensure continuous synchronization of local data repositories with the latest information available on the blockchain. *update* function intelligently checks for the last recorded data point and incrementally fetches new data until the current timestamp.

4.2.3 Data Filtering for Analysis

The *filtered_data* function serves this purpose by enabling the selection of data within predefined start and end dates. This temporal filtering is vital for conducting time-series analyses, backtesting trading strategies, and generating predictive models.

4.2 Framework for Amounts and Liquidity Calculations (liquidity.py, charts.py)

This section delves deeper into the mathematical intricacies of amount and liquidity calculations within Uniswap V3, particularly focusing on the Ethereum and USDC pool, converting mathematical formulas into Python coding.

4.3.1 Function Analysis (*get_amount0* and *get_amount1*)

- These functions calculate the amount of each token (ETH and USDC) in a liquidity position within a specific price range

Implementation

- Utilizes the constant product formula (*x * y = k)* in a modified form to accommodate Uniswap V3's concentrated liquidity feature. Calculates the amount of Token 0 (*get_amount0*) and Token 1 (*get_amount1()*) based on the square root values of the price range (sqrtA and sqrtB) and liquidity provided. Adjusts for token decimals to ensure precise calculation in terms of token units.

Square Root Price Model

- Central to Uniswap V3's mechanism is the use of square root prices (*sqrtPriceX96*) for liquidity calculations, offering higher capital efficiency. This approach allows for a more granular control over the price ranges in which liquidity is provided, directly impacting the amounts of tokens in the pool.

Practical Example

- Consider a scenario where a liquidity provider wants to determine the amount of ETH and USDC they would need to provide in a specific price range. Using get_amount0 and get_amount1, they can accurately calculate these amounts considering current pool dynamics and their desired price exposure.

4.3.2 Function Analysis (*get_liquidity0, get_liquidity1, and get_liquidity*):

These functions compute the total liquidity value of a position given the token amounts and the price range.

Implementation

- The functions calculate liquidity for each token separately (*get_liquidity0* for Token 0 and *get_liquidity1* for Token 1) and then find the minimum of the two to determine the position's effective liquidity.The calculations again rely on the square root price model, ensuring alignment with Uniswap V3’s core pricing mechanism.

Practical Example:

- A liquidity provider aiming to deposit ETH and USDC in a pool can use these functions to determine the optimal amount of each token to maximize their liquidity provision within their desired price range.This ensures that their capital is efficiently allocated, maximizing potential fee earnings while minimizing risk exposure.

4.3.3 Function Analysis (*charts1*):

- The chart1s function is designed to visualize and analyze the performance of a liquidity position within Uniswap V3 over a specified time frame. It aggregates key financial metrics to provide a comprehensive view of the investment's performance, including fee earnings, value changes, and impermanent loss.

Implementation

- *chart1* processes data acquired from the uniswap_backtest_ETH function, concentrating on vital metrics like fees earned and the valuation of liquidity positions in relation to market price movements. The function employs data resampling techniques to consolidate these metrics over regular intervals, such as daily, enhancing the clarity and understanding of trends and patterns. Furthermore, it dynamically computes various financial indicators, including the fee-to-value ratio and impermanent loss, offering insights into the overall return of the liquidity position compared to a simple hold strategy.

Practical Example

- A liquidity provider engaged in a range-bound strategy on the ETH-USDC pool, for instance, can utilize chart1s to monitor the progression of their position's value, accumulated fees, and potential impermanent loss. This tool aids

4.3.4 Framework for Backtesting Evaluation (backtest.py)

The *range_backtest* function in the *backtest.py* script embodies this concept, simulating the performance of liquidity provision strategies within specified price ranges.

- Inputs:
    - The function requires the start and end dates for the simulation period, an initial target investment (**target_0**), the chosen investment method, and a user-defined strategy (**my_strategy**).
- Process:
    - It begins by executing the specified strategy to determine the buy and sell positions within the given period.
    - For each buy position, it invokes the **uniswap_backtest_ETH** function to simulate a liquidity provision position from the opening to the closing of that position.
    - It dynamically adjusts the target investment based on the performance of each position.
- Performance Metrics:

- The function calculates the liquidity provider's return (**lp_return**) and the fee return (**fee_return**), providing a comprehensive view of the investment performance.

4.3.5 Detailed Analysis of the Computational Approach

- Compound Interest Method:
  - Under this method, the target investment is compounded after each position. This reflects a reinvestment strategy where the gains from one position are reinvested in subsequent positions.
  - The function tracks the cumulative fees earned (**cumulative_feeV**) and adjusts the hold target based on the strategy's performance.
- Data Aggregation and Output:
  - The function compiles an **important_sheets** DataFrame, aggregating key metrics from each simulated position. This includes daily resampling to align the data with regular time intervals.
  - It outputs this comprehensive dataset alongside the calculated fee return, offering a granular view of the strategy's performance.

4.4 Framework for Strategy Development in Backtesting (strategy.py)

This subsection outlines the general framework and approach for creating and implementing trading strategies within the backtesting environment, as demonstrated in the provided code samples.

The overarching framework for strategy development in backtesting can be segmented into several key steps, which are exemplified in the provided functions:

- Data Preparation and Indicator Calculation:
  - Each strategy starts by preparing the dataset (**df**), which typically includes historical price and volume data.
  - Key indicators are calculated based on historical data. These could include moving averages, standard deviations, or more complex indicators like Bollinger Bands and Average True Range (ATR).
- Defining Buy and Sell Conditions:
  - Strategies involve defining clear conditions for entering (buying) and exiting (selling) trades.
  - These conditions are based on the calculated indicators and can vary in complexity, from simple threshold-based rules to more intricate combinations of multiple indicators.
- Transaction Recording and Management:
  - Buy and sell transactions are tracked, often using dictionaries to map transaction dates with the associated buy or sell details.

    - Position management is crucial, with the code typically including logic to ensure the consistency of buy and sell actions.
- Iterative Approach:
    - The strategy often involves iterating through the dataset to evaluate the defined conditions at each time step.
    - This iterative approach allows for the dynamic adjustment of positions based on the evolving market conditions.

4.4.1 Practical Considerations in Strategy Development

- Customizability and Flexibility:
    - The framework should be flexible, allowing for easy modification and experimentation with different indicators and conditions.
    - Customizability is key to adapting the strategy to different assets or market conditions.
- Risk Management:
    - Incorporating risk management elements, such as stop losses or position sizing, is critical to ensure the strategy aligns with the risk tolerance of the user.
- Backtesting and Evaluation:
    - Once developed, strategies must be rigorously backtested over historical data to assess their effectiveness.
    - Evaluation metrics, such as return on investment, drawdown, and win/loss ratios, are crucial for assessing strategy performance.

5. Empirical Results

This analysis focuses on evaluating specific trading strategies applied within the Uniswap V3 framework. The backtesting period is from August 1, 2023, to November 14, 2023. Each strategy accounts for gas fees of approximately $15 per transaction, encompassing both the creation and dissolution of liquidity positions.

5.1 MiniStrategy1: Daily Rebalance with Unbounded Liquidity

- **Strategy Overview**: Implemented a daily rebalancing strategy with unbounded liquidity provision.
- **Results**: The outcome was notably poor, with consistent value loss. This indicates that the high frequency of rebalancing could not offset the gas fees, leading to net losses.
- **Conclusion**: High-frequency rebalancing in volatile markets, particularly with significant transaction costs, is unsustainable for profitability.

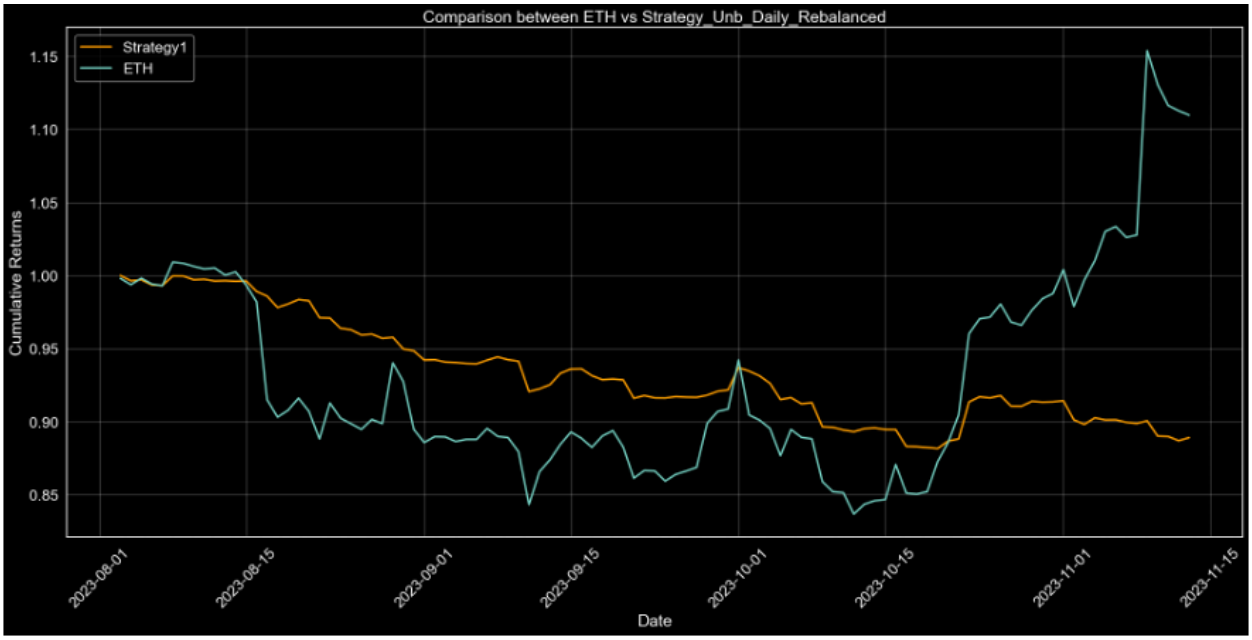

Figure 5. Rebalance Daily Strategy

5.2 MiniStrategy2: Benchmark Strategy

- **Strategy Overview**: A passive strategy involving unbounded liquidity provision over a two-year period.
- **Results**: Yielded an annualized return of about 3-4%, similar to the risk-free rate. This suggests that passive liquidity provision without managing impermanent loss is not highly profitable.

Figure 6. Two Year Unbounded Liquidity Provision

5.3 Strategy1: Bollinger Band Strategy

- **Implementation**:
  - The strategy calculates the Simple Moving Average (SMA) and standard deviation of the closing prices over a 14-day period.
  - Bollinger Bands are then formed by adding and subtracting twice the standard deviation from the SMA.
  - Buy signals are generated when the price crosses from below to above the lower band, indicating an oversold condition. Sell signals are initiated when the price crosses the upper band or meets predefined stop loss or profit targets.
- **Results and Conclusion**:
  - The returns closely followed the price movements of ETH, indicating effective mimicry of direct ETH investment with slightly improved risk management.

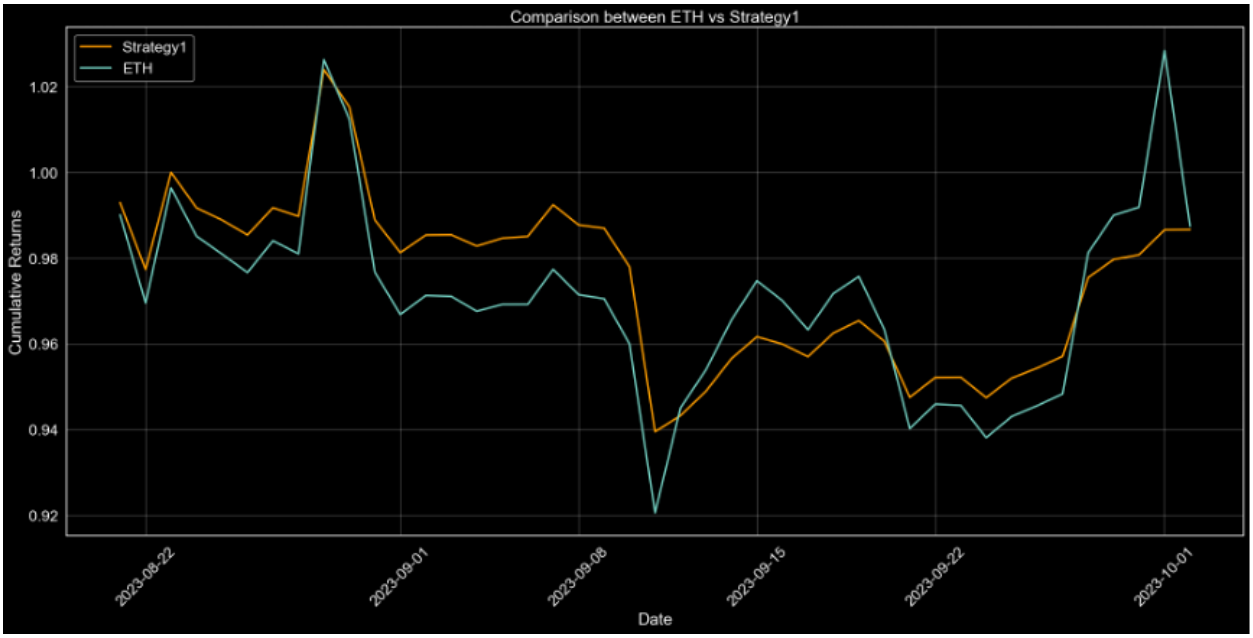

Figure 7. Bollinger Strategy

5.4 Strategy 2: Average True Range (ATR) Strategy

- **Implementation**:
  - The ATR strategy uses a 30-day window for both Bollinger Bands and ATR calculation. The ATR, a measure of market volatility, is calculated using the True Range (TR), which is the maximum of the current high minus low, current high minus previous close, and current low minus previous close.
  - The ATR value is then used to adjust the Bollinger Bands' width, by adding and subtracting a multiplier of the ATR to the SMA. This creates dynamic bands that adjust to market volatility.

- Buy signals are generated when the closing price moves from below to above the lower volatility-adjusted band, with sell signals triggered by the opposite movement or when the price hits dynamically set stop loss or profit levels, based on the ATR.

- **Results and Conclusion**:
  - This strategy demonstrated a significant ability to outperform direct ETH investment, effectively navigating market volatility and downturns. The use of ATR to adjust Bollinger Bands provided a more responsive approach to changing market conditions, allowing for better risk management and profit realization. Over a two-year testing period, the strategy avoided major downturns and capitalized on market movements.

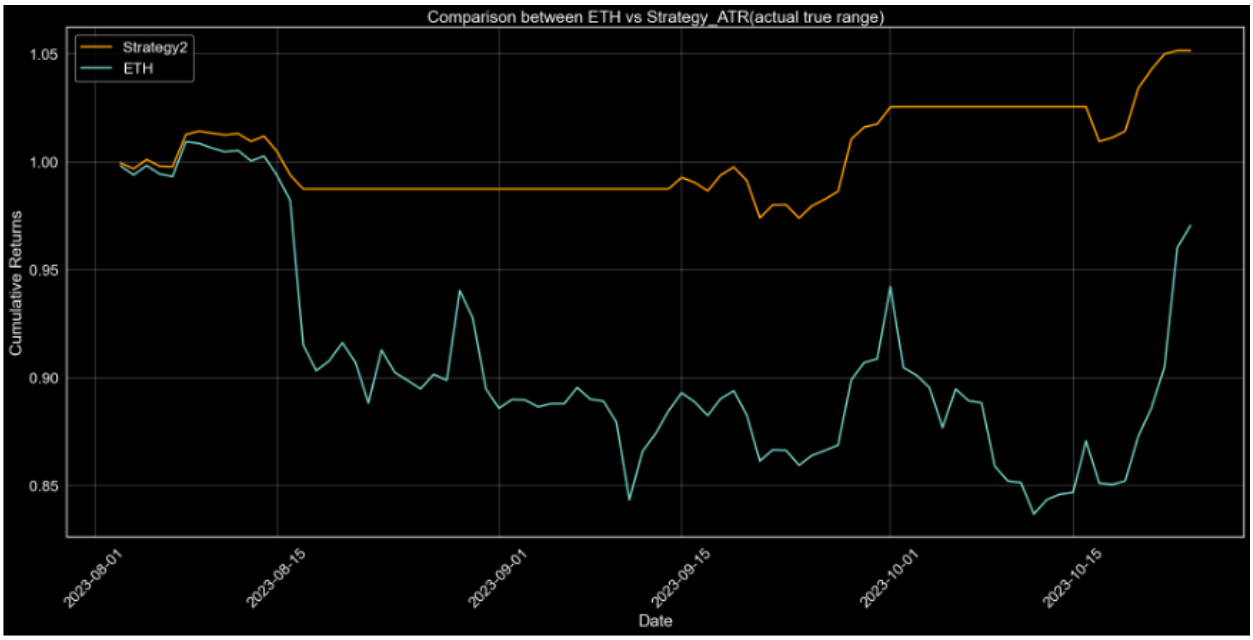

Figure 8. ATR Strategy with Backtesting Period

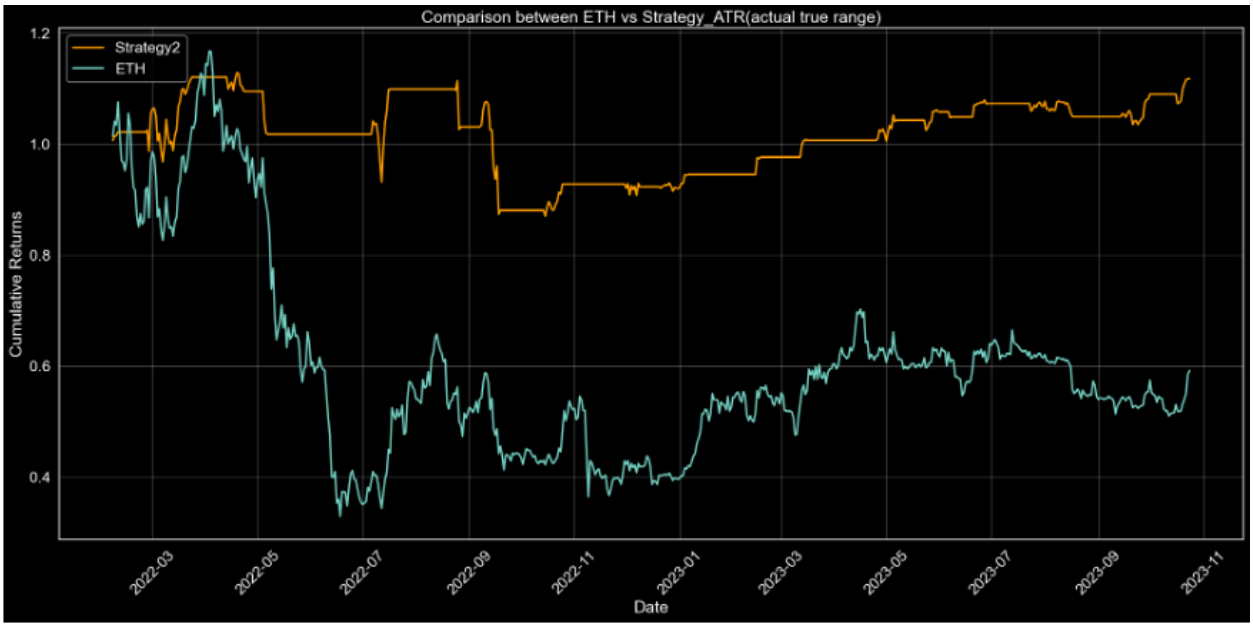

Figure 9. ATR Strategy in Two Years

5.5 Strategy 3: ARIMA Strategy

- **Implementation**:
  - This Strategy Divide each day's High minus Low by the previous day's close, creating a time series as illustrated in the following figure.

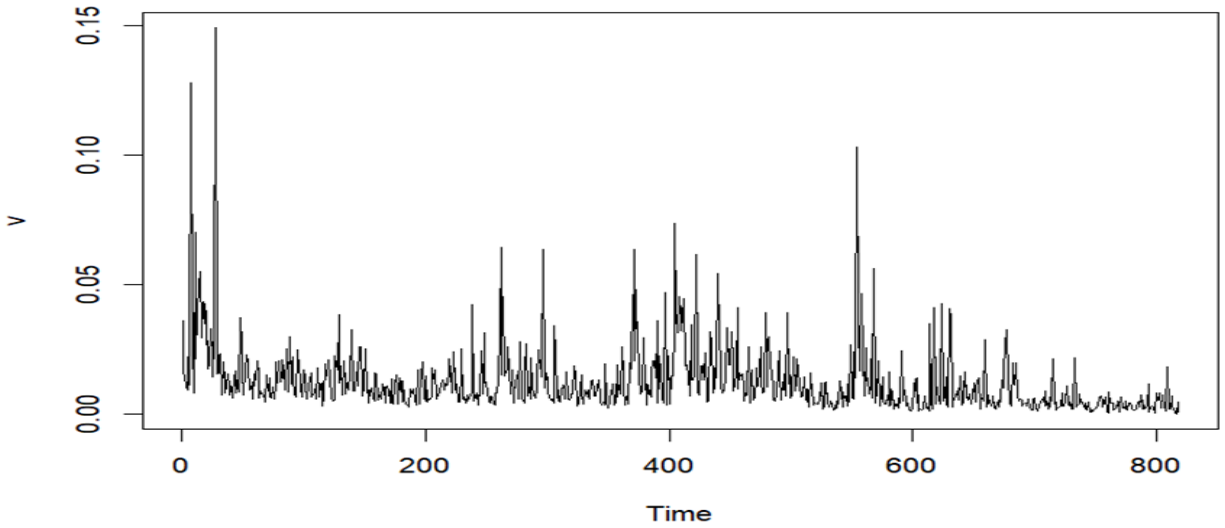

Figure 10. (High-low)/Previous Close

  - Then Used AIC as the standard for model selection, leading us to identify the optimal model as ARIMA(1,1,2).

- Conducted residual testing. Based on the Autocorrelation Function (ACF) plot of the residuals, it appears that the residuals do not exhibit autocorrelation. This suggests that the model fits the data well.

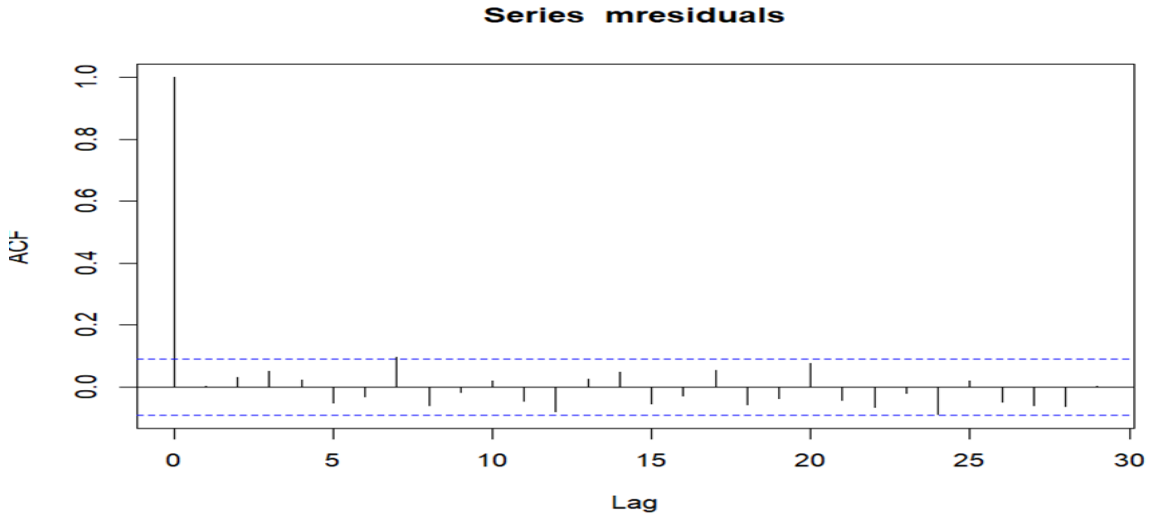

Figure 11. ACF of the residuals

- **Results and Conclusion**:
  - This strategy demonstrated a significant ability to outperform direct ETH investment, effectively reducing the risks of investment . The use of ARIMA to predict the price range is very accurate. Over a three-months testing period, the strategy demonstrates better outcomes in terms of both returns and volatility compared to simply holding ETH.

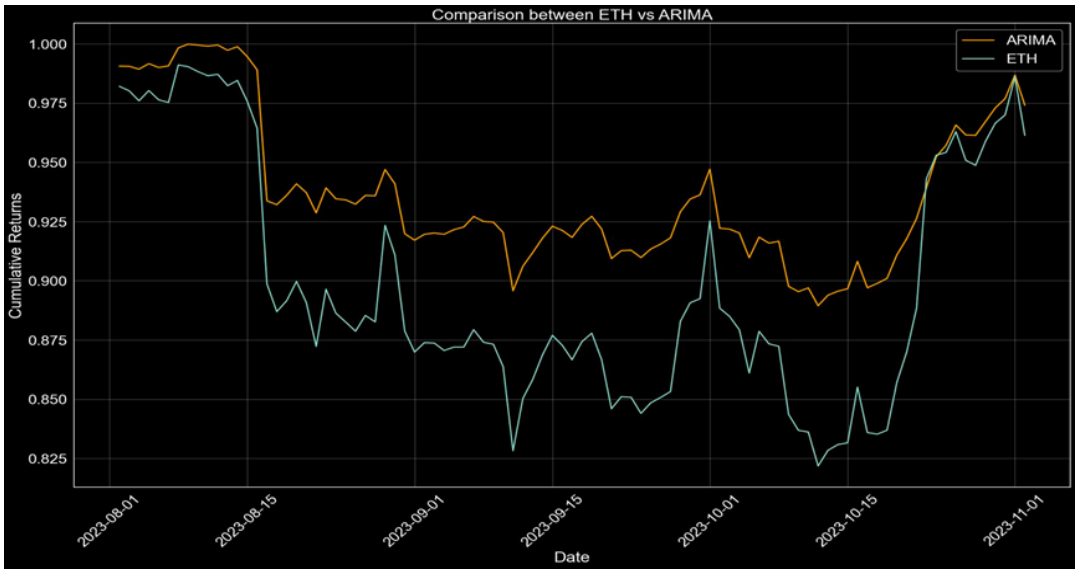

Figure 12. ARIMA Strategy with Backtesting Period

5.6 Strategy 4: Momentum and Mean-Reversion Strategy

- **Implementation**:
  - In Uniswap trading, there are individual and institutional traders. Individual traders generally have less comprehensive information and poorer investment techniques. When the market is dominated by them, Uniswap's volatility is likely to be relatively high, and mean reversion phenomena may occur. Conversely, when the market is dominated by institutional traders, the volatility should be lower, and momentum effects are more likely to be observed.
  - Based on this, when the Uniswap's recent volatility is greater than its historical volatility, we adopt a mean reversion strategy. This entails minting when the price is below the moving average and burning when the price is above the moving average. Conversely, when the recent volatility is lower than the historical volatility, we adopt a momentum strategy. Which means burning when the price is below the moving average and minting when the price is above the moving average.
- **Results and Conclusion**:
  - The backtesting results reveal the efficacy of this strategy in accurately forecasting both momentum and mean reversion phases. By strategically employing burning during periods of pronounced ETH price declines to mitigate losses, and simultaneous Minting during price upticks, this approach effectively curtailed volatility while securing substantial returns. This method not only anticipated market movements but also actively managed risk, resulting in a balanced and fruitful investment approach.

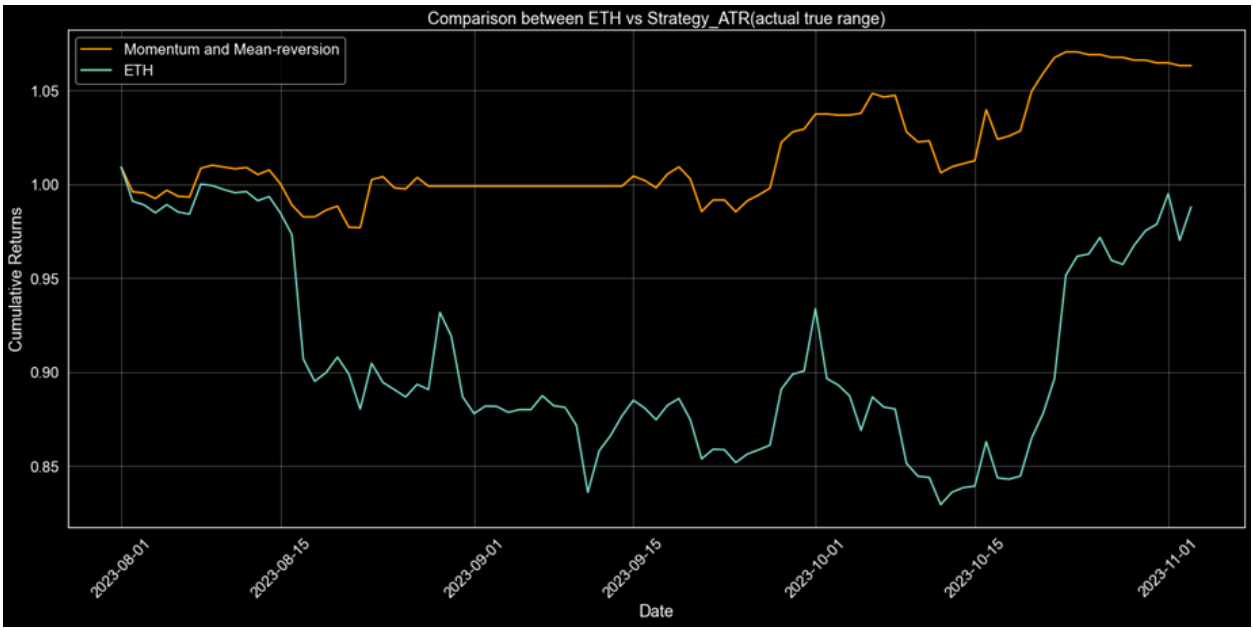

Figure 13. .Momentum and Mean-Reversion Strategy with Backtesting Period

5.7 Memory-Retaining Interval Strategy

To reiterate the goal described in the previous sections of this report, our main objective is to optimize the Liquidity Provisioning mechanism on Uniswap V3 by maximizing capital efficiency and minimizing the loss. The main source of our losses is due to impermanent loss, which occurs when the real current price tick comes outside of our liquidity provisioning bucket. In this event, depending on the side of the interval that the price crosses, all our tokens in both currencies are converted into only one type of tokens. If in this event the price of the token, that we no longer possess, increases, we accrue impermanent loss.

To counteract this, we need to consistently maintain the central tick price within our designated price range bucket. This approach aims to minimize the number of updates, thereby reducing the gas fees incurred from minting and burning when entering a new smart contract. To achieve this, we developed the Memory-Retaining Interval Strategy (MRIS). This strategy involves setting up two smaller intervals adjacent to the upper and lower ticks. These intervals ensure that if the current price tick falls within the critical range k, we then update our price range accordingly.

Now, our goal for the rest of this section will be to mathematically define k, corresponding update rule and test the strategy on the real data.

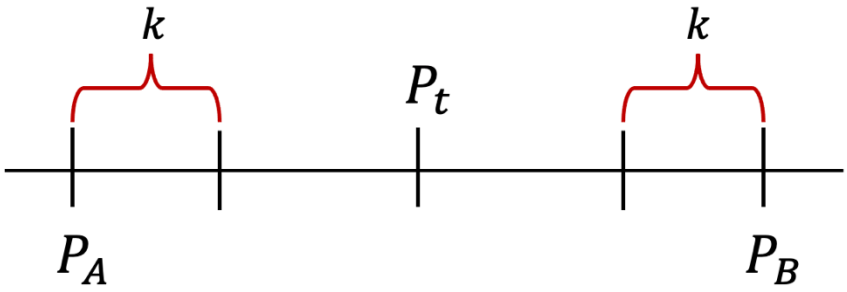

Figure 14. Interval k withing LP Price Range. P_A is the upper tick and $P_B$is our lower tick

5.7.1 Strategy Derivation

Previously we introduced k to be the length of the interval. It is obvious that the length of the interval should be proportional to some degree of price change. The more volatile the price, the larger the interval should be, and vice versa. To capture price fluctuations, we will use the trailing volatility metric (σ_n) over a certain span period (span) and scale our interval accordingly. The span will play a crucial role as the model's memory, particularly in determining the duration of the past that the model retains its understanding of price volatility characteristics. We determine the

length of the interval based on the volatility in a multiplicative manner, rather than an additive one. This means the following dependence:

$$interval\ size = k * \sigma_n \tag{18}$$

We employ a multiplicative approach, similar to a linear regression, where we condition the interval size (let's call it y) on the volatility measure in the following manner: $y = k * \sigma_n$ without intercept. If we were to consider k as an intercept in a linear relationship between interval size and volatility ($y = k + \beta\sigma_n$), it would imply a non-zero interval even when $\sigma_n = 0$. This in turn implies, that *if there are no price fluctuations, we would still need to account for price fluctuations caused by said volatility.* Since the statement is self-contradictory, we will multiply σ_n by k.

Additionally, we will scale the effect of the σ_n on the interval length by multiplying it by some weight α. This measure is taken to prevent our interval from exceeding half the length of the distance between two prices in cases of high volatility:

$$interval\ size = k * \alpha\sigma_n \tag{19}$$

Now, we want to identify the upper bound of α conditioned on the halfwidth of the price range. This means the following equation should be always satisfied:

$$\frac{1}{2}(P_A - P_B) > k * \alpha\sigma_n \tag{20}$$

If we solve for α:

$$\frac{1}{2k\sigma_n}(P_A - P_B) > \alpha \tag{21}$$

This gives us the upper bound of α under σ_n.

However, we want to find a dynamic upper bound within a certain memory span period and with respect to price evolution. The only element of the equation 21, that is defined by the system is the trailing volatility metric σ_n. That is to say, the upper bound changes with every σ_n, that we assume we do not control. In order to find the “true” upper bound over a certain time span, we need to minimize the RHS with respect to σ_n.

It becomes apparent that the LHS reaches its minimum when σ_n is at its maximum—since increasing the denominator results in a smaller fraction. Hence, our ”true” upper bound for some time span is:

$$\alpha < \frac{1}{2k\sigma_{n,\,max}}(P_A - P_B) \tag{22}$$

Since the LHS of the equation is only a fraction of the RHS, it is reasonable to say that there exists a weight multiplier, that equates both sides. Meaning for some value of w_i, such that $0 < w_i < 1$, the following holds:

$$\alpha = w_i \frac{1}{2k\sigma_{n,\,max}}(P_A - P_B) \tag{23}$$

Now we substitute equation α in equation 19 with the result given in the equation 23. Hence, we get:

$$interval\ size\ = k * \alpha\sigma_n = k\sigma_n w_i \frac{1}{2k\sigma_{n,\,max}}(P_A - P_B) \tag{24}$$

This allows us to remove k from our model and define the length of the interval at some time t:

$$interval\ size_t\ = w_i \frac{\sigma_{n,t}}{2\sigma_{n,\,max}}(P_A - P_B) \tag{25}$$

where

- $interval\ size_t$ is the interval size at time t.
- w_i is the weight parameter that determines the frequency of updates.
- $(P_A - P_B)$ is the set tick price range.
- $\sigma_{n,t}$ represents the exponentially weighted standard deviation, characterized by a decay determined over a span n. In this context, the weight of each observation decreases exponentially.
- $\sigma_{n,max}$ is the max volatility observed over the past n observations.

We refer to the ratio $\frac{\sigma_{n,t}}{\sigma_{n,\,max}}$ as the memory component of the model. This component enables the strategy to recall past volatility levels and accordingly adjust the interval. If the current exponentially weighted standard deviation is low, particularly in comparison to previously high volatility levels retained in the model's memory, the interval will be set less aggressively. This allows the central price tick greater mobility within the price range. Conversely, if the current volatility is high, the memory component approaches 1, thereby extending the interval length to its maximum (also defined by w_i) and prompting a more aggressive adjustment of the price range.

We update the price tick range as soon as the price get within the defined interval on either side:

If $P_t > P_{A,t} - interval$:

$$\begin{aligned} & P_{A,\,t+1} \\ &= P_{A,t} + interval\ size - (P_{A,t} - P_t) \\ &= P_{A,t} + interval\ size - P_{A,t} + P_t \\ &= P_t + interval\ size \\ P_{B,t+1} &= P_{A,t+1} - (P_{A,t} - P_{B,t}) \end{aligned} \tag{26}$$

If $P_t < P_{B,t} + interval$:

$$\begin{aligned} & P_{B,\,t+1} \\ &= P_{B,t} - interval\ size + (P_t - P_{B,t}) \\ &= P_{B,t} - interval\ size + P_t - P_{B,t} \\ &= P_t - interval\ size \\ P_{A,t+1} &= P_{A,t+1} + (P_{A,t} - P_{B,t}) \end{aligned} \tag{27}$$

However, the issue with this update strategy is that the tick price may land precisely on the interval's edge during the very next step if the volatility remains unchanged. We want to ensure that the tick price stays outside of the interval range. Therefore, we will introduce a small increment δ, to ensure that the tick price settles within the anticipated range.

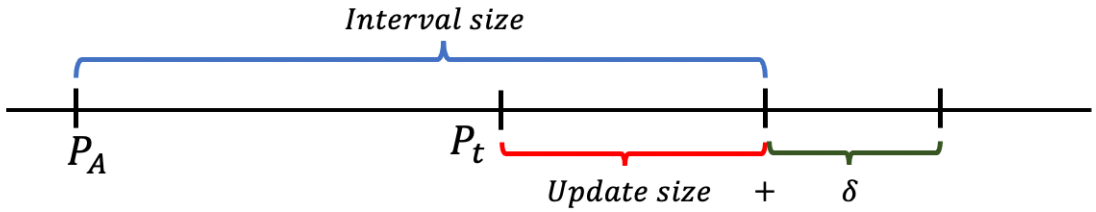

Figure 15. Price Range Update with δ in the upper halfwidth.

Since δ is located in between the interval the midpoint of the price range, it should satisfy the following equation:

$$\delta < \frac{1}{2}(P_A - P_B) - interval\ size \tag{28}$$

We can similarly indicate the upper bound of δ by substituting the interval size with the equation 8:

$$\delta < \frac{1}{2}(P_A - P_B)\left(1 - w_i \frac{\sigma_n}{\sigma_{n,\,max}}\right) \tag{29}$$

Using the same trick as with equation 23 for some w_δ, such that $0 < w_i < 1$:

$$\delta = w_\delta \frac{1}{2}(P_A - P_B)\left(1 - w_i \frac{\sigma_n}{\sigma_{n,\,max}}\right) \tag{30}$$

where

- $interval\ size_t$ is the interval size at time t.
- w_i is the weight parameter that determines the frequency of updates.
- w_δ is the weight parameter which determines the scale of delta.
- $(P_A - P_B)$ is the set tick price range.
- $\sigma_{n,t}$ represents the exponentially weighted standard deviation, characterized by a decay determined over a span n. In this context, the weight of each observation decreases exponentially.
- $\sigma_{n,max}$ is the max volatility observed over the past n observations.

This allows us to estimate the final update rule:

If $P_t > P_{A,t} - interval$:

$$P_{A,\,t+1} = P_t + \delta + interval \tag{31}$$
$$P_{B,t+1} = P_{A,t+1} - (P_{A,t} - P_{B,t})$$

If $P_t < P_{B,t} + interval$:

$$P_{B,\,t+1} = P_t - \delta - interval \tag{32}$$
$$P_{A,t+1} = P_{B,t+1} + (P_{A,t} - P_{B,t})$$

Now, we will test the performance of the strategy on the testing and training

5.7.2 Strategy Performance

In this subsection, we aim to evaluate the strategy's performance using various parameter sets applied to the same dataset. Our evaluation criteria include the number of updates per observation period, total return, standard deviation, fee income coverage[1], and the Sharpe ratio. The robustness of the model, under differing parameters, will be tested over a consistent span period of 5 days, totaling 105 days for the entire testing period. The descriptive statistics for both volatility measures are presented in table 1. The 5-day span for performance evaluation was chosen to capture short-term market dynamics while providing enough data points for statistical significance without being too granular. It allows for a comprehensive view of the strategy's behavior under various market conditions, offering insights into both the strategy's immediate execution and its stability over a near-term horizon.

[1] Calculated as a fee returns less impermanent loss, less gas fees over the testing period.

For the price range, we will examine the 500-tick range, as well as the 150 and 1000 the active ranges[2] based on distribution of ranges of mints and burns within the dataset. For each interval we will test the weights for δ and interval size at 0.4 and 0.9 making 4 possible combinations for each price range. The results of this analysis are presented in table 2. The trading amount given to the algorithm was 10,000 USD.

Metric	EW Volatility	Max Volatility
count	100.00	100.00
mean	215.01	282.92
std	134.56	153.46
min	35.25	57.49
25%	123.71	180.47
50%	187.60	245.75
75%	246.06	333.03
max	654.28	654.28

Table 1. Descriptive Statistics for Exponentially Weighted and Maximum Volatilities over a testing period with a span of 5 days

Price Range	Interval Weight	Delta Weight	Number of Updates	Total Return	Standard Deviation	Fee Coverage	Sharpe Ratio
150	0.4	0.4	66	133.14%	31.57%	3464.86	4.22
	0.9	0.4	82	174.13%	31.42%	5622.28	5.54
	0.4	0.9	65	149.73%	32.10%	3832.48	4.66
	0.9	0.9	80	176%	31.52%	5515.64	5.58
500	0.4	0.4	34	34.83%	26.97%	-756.2	1.29
	0.9	0.4	53	60.09%	28.97%	-341.4	2.07
	0.4	0.9	31	50.95%	28.87%	-650.8	1.76
	0.9	0.9	54	67.57%	29.27%	-177.14	2.31
1000	0.4	0.4	19	12.30%	22.85%	-734.44	0.54
	0.9	0.4	36	27.59%	26.30%	-871.478	1.05
	0.4	0.9	15	18.05%	24.32%	-582.408	0.74
	0.9	0.9	35	29.66%	26.88%	-955.96	1.10

Table 2. Algorithm Performance under different sets of parameters based on the described previously back tester.

[2] In the future it is also possible to augment the strategy for the dynamically changing price range.

According to the back-testing results, the most effective strategy is characterized by a tight interval of 150, with both weights set at 0.9. A tighter range results in a higher concentration of liquidity within that range, leading to a greater accumulation of fees. These fees are sufficient to more than compensate for the impermanent loss and the incurred gas fees. In contrast, the returns from strategies with larger interval sizes are heavily dependent on the performance of the underlying ETH price. This is because the fees generated in these cases are inadequate to offset the losses caused by both gas fees and impermanent loss.

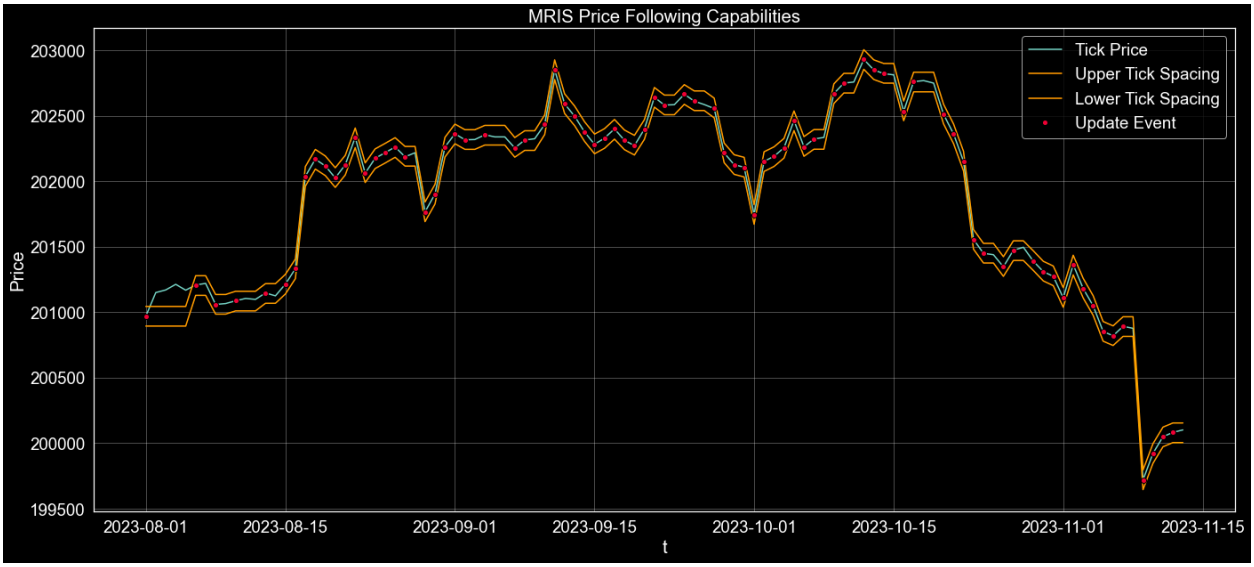

Figure 16. Price Following Capabilities of the MRIS on the testing period

Figure 16 showcases the price tracking capabilities of the best-performing strategy. This strategy demonstrates an effective ability to adjust its range dynamically, ensuring that the central price tick remains within the targeted range. At the onset of the observed period, a minor offset is noticeable. This initial discrepancy can be attributed to the model requiring a brief period to adjust and accumulate sufficient memory for optimal functioning. During this adjustment phase, the model calibrates its memory component based on incoming market data, which initially may result in less precise alignment with the central price tick. However, as the model continues to gather and process data, its accuracy in maintaining the central price tick within the desired range improves, reflecting its robust adaptive capability in response to market fluctuations.

Strategy's PnL over the testing period against the ETH price can be seen on Figure 17. We can see a steady increase in cumulative returns mostly due to compound reinvesting of fees received over the testing period.

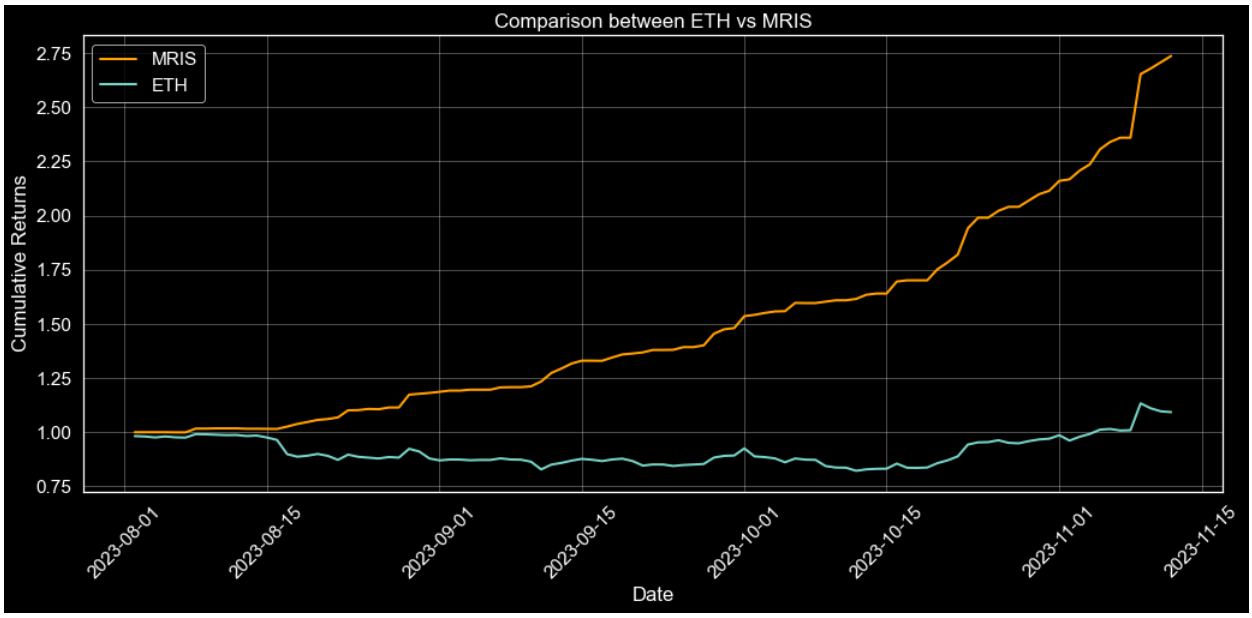

Figure 17. MRIS PnL against Buy and Hold ETH.

5.8 Reinforcement Learning

In this section we are going to introduce our final strategy based on Reinforcement Learning Framework. Given that the liquidity provision problem is structured as a Markov decision process with a discrete action space, we will implement a Deep Q-Learning agent to achieve an adaptive strategy for liquidity provisioning.

We formulate the optimal liquidity provision problem as a policy search problem in a Markov decision process denoted by $\{S, A, P, r\}$. S and A are discrete state and action space, P denotes state transition probability of the dynamic environment $P(s', r, s, a) = \Pr\{S_{t+1} = s', r_t = r | S_t = s, a_t = a\}$. r represents a reward function and $r(s_t, a_t)$ is the reward received by the acting agent at state s_t for performing action a_t. Agent's action is determined by a policy π, where π must satisfy $\sum_{a_t \in A} \pi(a_t|s_t) = 1$. $\pi(a_t|s_t)$ represents the probability of performing action a_t at state s_t.

The objective of the optimal policy is to maximize the discounted reward $R_t = \sum_{t=1}^{\infty} \gamma^t r_t$, where γ is a discount factor such that $0 < \gamma < 1$. The discount factor is present to reinforce the idea that present rewards are more important, compared to future rewards at a given time t. State value function is the expected reward under some policy π, defined as $V^{\pi}(s) = \mathbb{E}_{\pi,P}[R_t|S_t = s]$. The Q function is defined as a realized expectation of the value function under state s and action a: $Q^{\pi}(s, a) = \mathbb{E}_{\pi,P}[R_t|S_t = s, A_t = a]$. Optimal Q function is defined as a maximum over all possible policies, performed by the agent:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\pi,P}[R_t|S_t = s, A_t = a] \tag{33}$$

One way to estimate $Q^*(s,a)$ is by using Bellman optimality equation to do an interactive update:

$$Q^*(s,a) = \sum_{s',r} P(s',r,s,a)[r + \max_{a'} Q^*(s,a)] \tag{34}$$

Deep Q-Learning is an advanced technique employed in the realm of reinforcement learning, rather than supervised learning, to deduce the optimal policy for a given decision-making problem. This method relies on approximating the Bellman Optimality Equations, utilizing an initial set of exploratory trajectories generated by an ϵ-greedy policy. This policy strikes a balance between exploration (choosing random actions) and exploitation (choosing actions based on known information) to navigate the state space effectively. As the agent interacts with the environment, it maps actions to states and the subsequent rewards, creating a rich dataset of experiences. These experiences are then stored in a replay buffer, which serves as the initial training set. The replay buffer's purpose is to break the temporal correlations between the sequences of observations, thereby facilitating a more stable and convergent learning process. By sampling from this buffer, the deep neural network—often referred to as the Q-network—can be trained iteratively to estimate the quality of state-action pairs, guiding the agent towards the optimal policy through incremental learning and continuous refinement.

5.8.1 State Space Definition

We represent our state as $S = \{P_{A,t},\ P_{B,t},\ L_t,\ P_t,\ LD_t\}$. This set comprises the upper tick price ($P_{A,t}$), the lower tick price ($P_{B,t}$), the liquidity position (L_t), the central tick price (P_t), and the encoded liquidity distribution (LD_t). These elements provide a comprehensive view of the conditions at UniswapV3 at any given time step t, enabling the RL agent to assess the current state effectively.

The upper and lower tick prices define the boundaries of the agent's price range. The central tick price represents the current market price, around which the agent must strategize its position. The liquidity position is indicative of the agent's current stake or investment in the UniswapV3, which can affect its risk and reward potential. Lastly, the liquidity distribution reflects how the agent's investment is spread across different prices against other's liquidity positions, affecting the potential returns and the risk of slippage. Together, these components form a multi-dimensional space where the RL agent navigates and learns the optimal policy for maximizing returns while managing risks through interactions with the environment.

5.8.2 Action Space Definition

For this problem, we will discretize our action space in a manner similar to the MRIS strategy. The agent will have access to a fixed interval length within which it can perform interval movements. The action set A is defined as {-500, …, 0, …, 500}, with each action spaced at one hundred tick intervals. This discretization process simplifies a potentially infinite set of actions into a manageable, finite set, thus allowing the agent to make decisions more efficiently. An action

of 0 indicates no change, meaning the agent will maintain the current interval without any movement.

5.8.3 Reward Definition

We set the reward function of our agent to be the change in its position value:

$$r_t = -c^* 1_{\{A \neq 0\}} + Fee_t + \Delta V_t, \tag{34}$$

where

- r_t is the reward function at time t.
- c is the average gas fee for existing and entering a new smart contract.
- $1_{\{A \neq 0\}}$ is an indicator function, which tracks whether agent performed and action.
- Fee_t is the fee earned at time t.
- ΔV_t is the change in agent holdings.

ΔV_t is estimated using the following equation:

$$\Delta V_t = p_{t+1} x_{t+1} + y_{t+1} - (p_t x_t + y_t), \tag{35}$$

where

- p_t is the central price at time t.
- x_t is the number of tokens 0 at time t.
- y_t is the number of tokens 1 at time t.

5.8.4 Model Assumptions and Results

In the following experiments we will be adopting the following assumptions:

- **Assumption 1:** The gas fee for each liquidity reallocation, which encompasses the sum of the gas fees to withdraw liquidity, adjust holdings, and invest liquidity again, is assumed to be a flat fee.
- **Assumption 2:** Dynamic trading applying the rebalancing strategy is conducted without friction.
- **Assumption 3:** Slippage is neglected when the agent adjusts holdings after each liquidity withdrawal.
- **Assumption 4:** The actions taken by the agent have no effect on the behaviors of other players in the market.

The agent's episode performance is illustrated in Figure 18 and the agent's PnL is presented in Figure 19. As depicted, the agent demonstrated a consistent ability to earn 395 USD during the

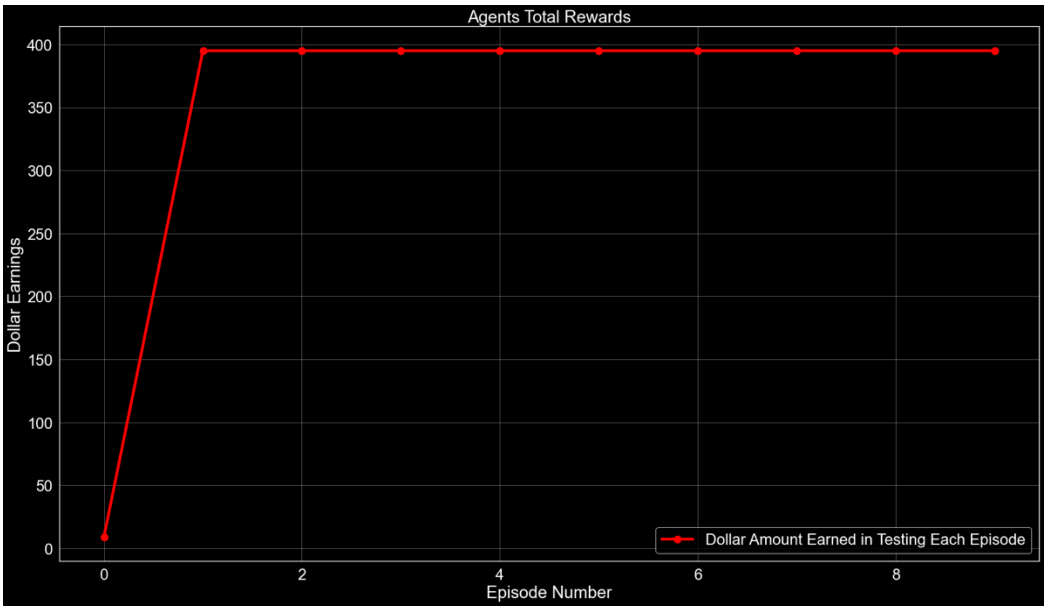

Figure 18. RL Strategy Returns Based on 8 Testing Episodes.

testing period, accompanied by a stable standard deviation. It's worth noting that the relatively low earnings can be attributed primarily to the agent's limited training duration. To optimize its performance, further enhancements and refinements are required.

The agent's capability to consistently earn 395 USD is indicative of its reliability in the given task. However, this achievement should be considered in the context of the training period, which may

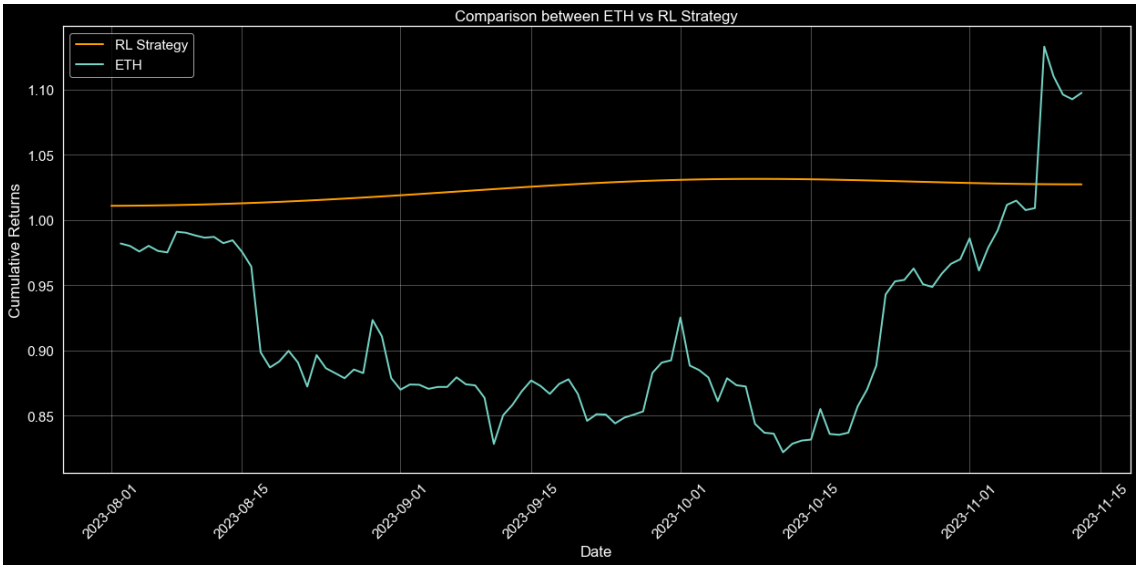

Figure 19. RL Strategy PnL against Buy and Hold ETH.

not have provided the agent with sufficient exposure to diverse scenarios and data. As a result, there is room for improvement in its performance.

5.8 Results comparison between all models

The performance of all strategies can be seen from the table 3. The metrics evaluated include Impermanent Loss, Fee Collected, Total Return, and the Sharpe Ratio, which is a measure of risk-adjusted return.

The Memory-Retaining Interval Strategy outperforms other strategies significantly, with the highest Fee Collected at 8412.53 and an impressive Total Return of 176%. The Sharpe Ratio for this strategy stands at 5.58, indicating an exceptionally high level of return per unit of risk taken. Conversely, this strategy also incurs a substantial Impermanent Loss of -1616.88, which warrants attention in the context of its overall profitability.

In contrast, the ARIMA strategy shows a negative Total Return at -1.92% and a low Sharpe Ratio of -0.49, suggesting an unfavorable risk-adjusted return profile despite collecting fees of 1572.60. The Bollinger Band strategy, while also recording a negative Total Return, shows a marginally better risk profile than the ARIMA strategy. Strategies such as ATR with BB and Momentum and Mean-Reversion display positive Total Returns and Sharpe Ratios above 1, indicating more favorable risk-adjusted returns. Lastly, the Reinforcement Learning Strategy demonstrates the least Impermanent Loss, a modest Total Return of 0.3%, and a Sharpe Ratio of 1.01, suggesting a balanced strategy with a stable return profile when considering risk.

Strategy	Impermanent Loss	Fee Collected	Total Return	Sharpe Ratio
Bollinger Band	-352.11	299.56	-0.75%	-0.46
ATR with BB	-795.25	443.95	3.76%	1.27
ARIMA	-1642.54	1572.60	-1.92%	-0.49
Momentum and Mean-Reversion	-1411.88	672.61	6.03%	1.46
Memory-Retaining Interval Strategy	-1616.88	8412.53	176%	5.58
Reinforcement Learning Strategy	-987.23	1201.12	0.3%	1.01

Table 3. Comparison between Performances of all Strategies

6. Conclusion

In conclusion, the Memory-Retaining Interval Strategy emerges as the most profitable approach among the various strategies assessed in the report, yielding the highest fee and return. Its robust performance, as indicated by the Sharpe Ratio, suggests that the returns significantly outweigh the risks taken, making it an attractive strategy for traders seeking substantial gains. However, the notable impermanent loss associated with this strategy should not be overlooked, as it points to potential volatility and the necessity for careful risk management.

An overview of Uniswap v3 within the context of this analysis suggests that while it offers innovative mechanisms for liquidity provision and fee generation, traders and liquidity providers must carefully navigate the risks of impermanent loss. The performance of the trading strategies on Uniswap v3, especially with the introduction of concentrated liquidity, may present both opportunities and challenges. The platform's complex fee structure and the potential for high returns warrant further examination to optimize strategy selection and capital allocation for achieving desired investment outcomes.

7. Literature and References

1) Adams, H., Zinsmeister, N., Salem, M., Keefer, R., & Robinson, D. (2021, March). Uniswap v3 Core.
2) Fritsch, R. (2021). Concentrated Liquidity in Automated Market Makers.
3) Neuder, M., Rao, R., Moroz, D. J., & Parkes, D. C. (2021). Strategic Liquidity Provision in Uniswap v3. arXiv preprint arXiv:2106.12033.
4) Elst, A. (2021). Liquidity Math in Uniswap V3.
5) Adams, H., Zinsmeister, N., Salem, M., Keefer, R., & Robinson, D. (2021). Uniswap V3 Core.
6) Erins, P. (2021). Impermanent Loss in Uniswap V3. Retrieved from https://medium.com/auditless/impermanent-loss-in-uniswap-v3-6c7161d3b445.
7) Zhang, H., Chen, X., & Yang, L. F. (2023). Adaptive Liquidity Provision in Uniswap V3 with Deep Reinforcement Learning. arXiv preprint arXiv:2309.10129v1 [cs.AI].
8) Zumbach, Y. (Year). Reinforcement Learning for Blockchain-Based Trading. Semester Thesis, Distributed Computing Group, Computer Engineering and Networks Laboratory, ETH Zurich.
9) Hsu, A., & Sreekumar, A. (2023). A Reinforcement Learning Approach to Optimal Liquidity Provision in Uniswap v3. University of California, Los Angeles.
10) J. Clark, “The Replicating Portfolio of a Constant Product Market with Bounded Liquidity,” Available at SSRN, 2021.
11) J. Yin and M. Ren, “On Liquidity Mining for Uniswap v3,” arXiv preprint arXiv:2108.05800, 2021.

For replication and further research, the code utilized in this study is available in the following GitHub Repository: https://github.com/yuyuCornell/UniswapV3DarkForest.git.