Intelligent Liquidity Provisioning Framework V1

Exploring Advanced Strategies in Uniswap V3 Liquidity Provisioning with Reinforcement Learning and Agent-Based Modeling

Abstract

Liquidity provisioning in Uniswap V3 presents a stochastic optimal control problem with a well-defined utility function to maximize. This article introduces an innovative framework for intelligent liquidity provisioning, utilizing a combination of agent-based modeling and reinforcement learning. Our framework provides a robust and adaptive solution for optimizing liquidity provisioning strategies. The Uniswap V3 model mimics real-world market conditions, while the agent-based model (ABM) creates an environment for simulating

agent interactions with Uniswap V3 pools. The reinforcement learning agent, trained using deep deterministic policy gradients (DDPG), learns optimal strategies, showcasing the potential of machine learning in enhancing DeFi participation. This approach aims to improve liquidity providers' profitability and understanding of CFMM markets.

Introduction

In my previous article on market making [Market Making Mechanics and Strategies], we explored the mechanics and strategies of market making in traditional financial markets. Building upon those insights, this article introduces an innovative framework for intelligent liquidity provisioning in the context of Uniswap V3. As mentioned in our prior research, our goal was to extend our understanding of market dynamics and liquidity management in decentralized finance (DeFi), specifically through the development of the Intelligent Liquidity Provisioning Framework.

Decentralized finance (DeFi) has undergone remarkable growth, introducing innovative financial products and services accessible to a global audience. Uniswap V3, at the forefront of this innovation, has revolutionized liquidity provisioning with its concentrated liquidity feature. However, this advancement brings forth complex decision-making challenges for liquidity providers. This article introduces a comprehensive framework designed to address these challenges, offering a simulated environment for studying and optimizing liquidity provisioning strategies.

Our framework comprises three key components: the Uniswap V3 model, an agent-based model (ABM), and a reinforcement learning agent. The Uniswap V3 model provides a representation of the pool, enabling the deployment and interaction with tokens and pools. The ABM introduces complexity by simulating agent interactions and market dynamics, creating a rich environment for strategy evaluation. The reinforcement learning agent, operating within this environment, adopts a deep deterministic policy gradient approach to learn and

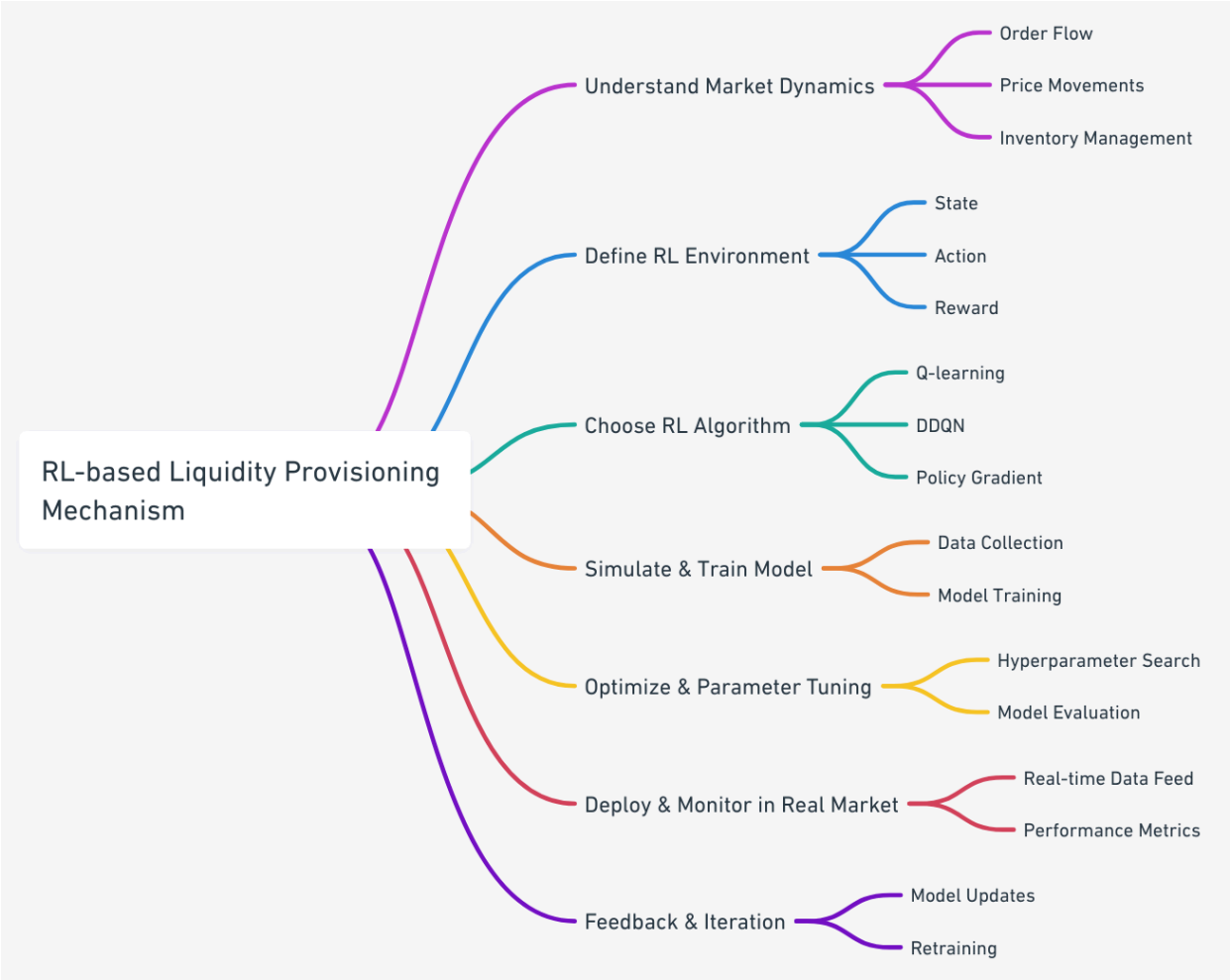
adapt strategies, aiming for optimal performance in liquidity provisioning.

ILP Framework

This research aims to develop an intelligent liquidity provisioning (ILP) mechanism using reinforcement learning (RL) to autonomously manage and optimize liquidity within the Uniswap V3 environment. The mechanism seeks to maximize the utility function, considering fees earned, impermanent loss, and other metrics based on liquidity providers' preferences while adapting to the complex dynamics of the CFMM market.

Intelligent Liquidity Provisioning Framework

In the RL framework, the liquidity provisioning problem is formulated as a Markov Decision Process (MDP). The MDP consists of states, actions and rewards.



ILP Mechanism

States: States represent the current market conditions, including asset prices, trading volumes, and other relevant variables.

Actions: Actions correspond to the decisions made by the liquidity provider, such as adjusting liquidity allocations, rebalancing portfolios etc.

Rewards: Rewards quantify the desirability of the outcomes based on the liquidity provider's objective function, preferences, and constraints. The rewards can be positive for desirable outcomes (e.g., high returns) and negative for undesirable outcomes (e.g., high risk or underperformance).

Objective Function: The objective function represents the liquidity provider's desired outcome, which can be a combination of factors like maximizing returns, minimizing risks, or achieving a specific trade-off between the two. Constraints can include limitations on liquidity allocations, capital utilization, risk tolerance levels, or other restrictions defined by the liquidity provider.

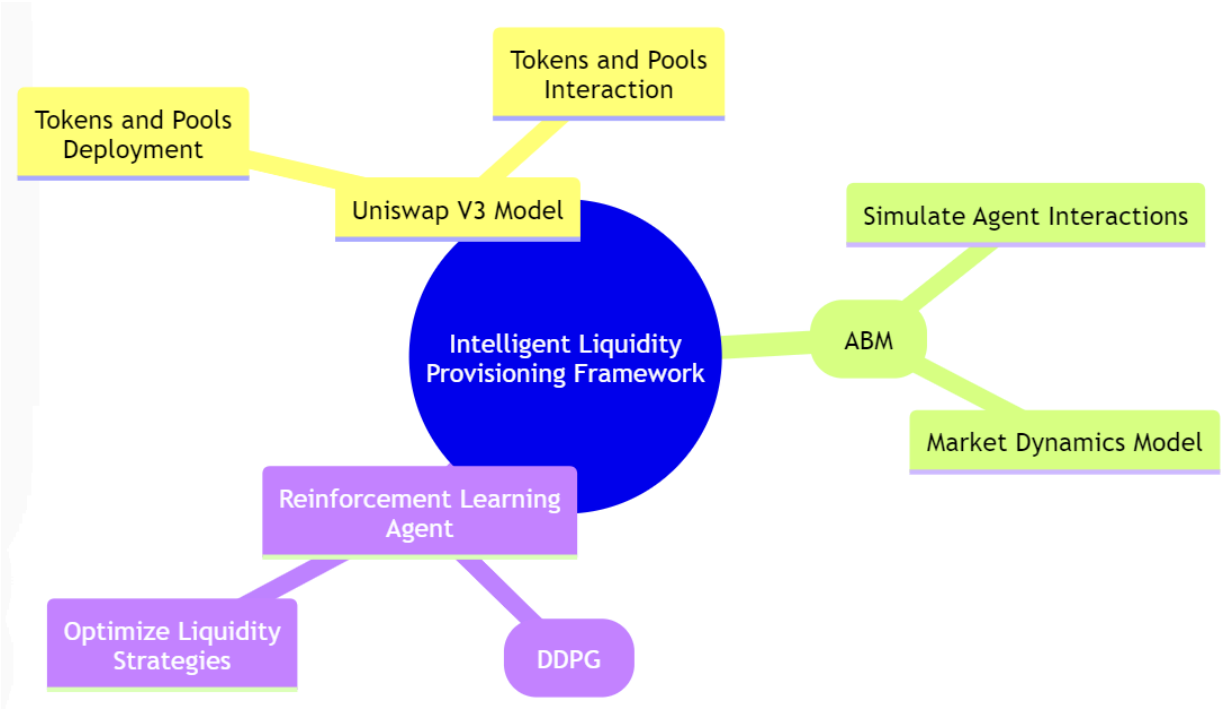
RL training is an iterative process where the agent continuously updates its policy based on feedback. The agent learns from its experiences and refines its decision-making over time, gradually converging to more optimal liquidity provisioning strategies.

Once the RL agent has been trained, it can be tested and evaluated using historical data or simulated environments to assess its performance against the liquidity provider's objective function and constraints. The agent's performance can be measured using metrics like returns, risk measures, or other relevant performance indicators.

By applying RL algorithm, the liquidity provisioning mechanism can learn and adapt to changing market conditions, identify optimal liquidity provision strategies, and balance constraints and preferences specified by the liquidity provider. RL enables the mechanism to find solutions that maximize the liquidity provider's objective function, considering various trade-offs and constraints in an autonomous and dynamic manner.

Components of Intelligent Liquidity Provisioning Framework

The framework comprises three major components:



Framework Components

UniswapV3 Model

The Uniswap V3 model implemented in Python offers a detailed and functional simulation of the Uniswap V3 protocol, capturing its nuanced mechanics and providing users with a comprehensive toolset for interacting with the protocol. The UniswapV3_Model class handles the deployment of tokens and pools, initializes pools, and provides an interface for pool actions and pool state retrieval.

Overview

The Uniswap Model serves as the foundation of the Intelligent Liquidity Provisioning Framework, encapsulating the core mechanics of Uniswap V3. It leverages compiled smart contracts from Uniswap's V3-Core, deployed in a local Ganache environment using brownie, to create a realistic and interactive simulation.

Contract Compilation and Deployment

The framework integrates with Brownie, a Python-based development and testing framework for smart contracts, to compile and deploy the Uniswap V3 smart contracts. These contracts are then deployed to a local Ganache environment, providing a sandbox for testing and development. This setup ensures that users can interact with the Uniswap environment without the need for real assets or network transactions, fostering a safe and controlled experimentation space.

Intelligent Liquidity Provisioning Framework V2—Explanation

Abstract

Liquidity provisioning in decentralized finance (DeFi) platforms like Uniswap V3 is a complex challenge, best described as a stochastic optimal control problem. The goal is to devise a strategy that maximizes a well-defined utility function, reflecting the preferences and objectives of liquidity providers (LPs).

Introduction

This third article in the market making series builds on a previously developed framework for optimizing liquidity provisioning in Uniswap V3, further enhancing it by addressing Previous version's shortcomings.

This article introduces a framework for intelligent liquidity provisioning in Uniswap V3, addressing it as a stochastic optimal control problem. It combines agent-based modeling with reinforcement learning to optimize liquidity strategies. The Uniswap V3 model simulates real market conditions, and the agent-based model (ABM) allows for interaction simulations in Uniswap V3 pools. A reinforcement learning agent trained with deep deterministic policy gradients (DDPG) and proximal policy optimization(PPO)identifies optimal strategies. This method seeks to enhance liquidity providers' profits while mitigating risks associated with liquidity provisioning.

Agent Environment

The agent's environment is simulated using local instances of Uniswap V3 pools, leveraging agent-based modeling to encapsulate the dynamic nature of these markets. This model includes various actors such as traders (noise, informed, arbitrageurs, MEV) and LPs (retail, institutional), whose behaviors are delineated through policies based on historical data analysis, as found in ***model_scripts/policies.py***.

State Space

The state space includes key metrics of the pool's condition: liquidity, current price, and fee growth in both directions. These metrics guide the RL agent's decisions to optimize liquidity provisioning by considering factors such as slippage, cost, and trading volume potential.

Liquidity: Indicates the total liquidity depth, informing the agent about potential slippage and swap costs. Higher liquidity depth reduces the risk of impermanent loss, attracting more trade and fees.

Current Price: Represents the pool's current tick, guiding liquidity addition around the prevailing market price.

Fee Growth: FeeGrowth0 and FeeGrowth1 indicate the volume of swaps in one direction, aiding in decision-making about the market's direction.

Action Space

The agent's action space includes possible actions it can take, specifically the setting of lower and upper price bounds for

liquidity provisioning within predefined limits to prevent drastic actions.

Action in our framework consists of a tuple (price_lower, price_upper), which is scaled back to the original scale of the pool using lower and upper bounds defined for the action.

Customized Reward Function

The reward function is intricately designed to cater to the unique preferences and objectives of liquidity providers (LPs). This function is a critical component, as it directly influences the agent's behavior by rewarding or penalizing actions based on their outcomes. The reward is calculated as the difference between fees earned and the impermanent loss (IL) incurred, adjusted for each time step. To personalize this function according to LPs' preferences, we introduce weightings to fees

earned and IL, which reflect the LPs' individual priorities. These priorities include:

Risk Tolerance: LPs can specify their willingness to accept risk, influencing the reward function to either favor riskier strategies with potentially higher rewards or more conservative strategies to minimize loss.

Profit Motivation: This factor determines how aggressively the reward function should pursue fee generation, balancing the quest for profits against the risk of impermanent loss.

Cost Sensitivity: Reflects the LP's concern over transaction costs, including gas fees, which can erode profitability.

Liquidity Utilization: Encourages strategies that optimize the use of provided liquidity, aiming for efficient capital deployment.

By adjusting these weights, the reward function dynamically aligns with the LP's specific investment strategy and risk profile, offering a tailored approach to liquidity provisioning in DeFi markets.

RL Algorithms

The ILP framework utilizes advanced RL algorithms to train agents for optimal liquidity provisioning strategies in DeFi platforms like Uniswap V3. These algorithms are selected for their efficiency, stability, and capability to handle the complex, dynamic nature of financial markets.

Deep Deterministic Policy Gradient (DDPG)

DDPG is an algorithm that combines the benefits of both policy gradient methods and Q-learning. It is designed to work in

continuous action spaces, making it particularly suitable for the ILP framework, where the action space involves setting precise liquidity ranges.

How DDPG Works: DDPG uses a deterministic policy to select actions, and a separate critic network to evaluate the action's potential reward. This separation allows for efficient learning by directly optimizing the policy based on the critic's feedback, without the need for exhaustive exploration of the action space.

Application in ILP: In the context of ILP, DDPG helps the agent learn the most effective positions for liquidity provision by evaluating the potential reward (fees minus impermanent loss) of different price ranges. This enables the agent to adapt its strategy to maximize returns or minimize losses based on historical and current market data.

Proximal Policy Optimization (PPO)

PPO is another advanced RL algorithm known for its stability and robustness, particularly in environments with high uncertainty. It is a policy gradient method that improves upon previous algorithms by using a novel objective function, making it more effective at balancing exploration and exploitation.

How PPO Works: PPO optimizes a clipped version of the objective function, which prevents the policy from changing too drastically at each update. This leads to more stable and reliable learning, as the algorithm avoids making large, potentially harmful updates to the policy.

Application in ILP: For ILP, PPO's stability and reliability are crucial. The algorithm allows the agent to incrementally improve its liquidity provisioning strategy, ensuring that it can adapt to

market changes without the risk of sudden, unprofitable shifts in policy. This gradual improvement process is particularly beneficial in the volatile DeFi market, where conditions can change rapidly.

ABM Policies

The ABM policies within our ILP framework are designed to simulate the complex interactions and behaviors of various market participants within the DeFi ecosystem. These policies enable our RL agent to navigate and respond to a dynamic market environment realistically. We employ two main approaches to model these policies**:**

1. **Mathematical Modeling:** This approach uses mathematical equations to describe the behavior of agents based on internal and external factors. Internal

factors include slippage, liquidity depth, and asset volatility, while external factors encompass broader market indicators like Bitcoin price and interest rates. This method allows for precise, rule-based representation of agent behaviors.

2. **Statistical Modeling:** Leveraging historical market data, this supervised learning approach trains models to predict trader actions based on a set of features. These features encapsulate both internal market conditions and external economic indicators, with trader actions serving as labels. This method enables the framework to adapt its strategies based on observed market patterns and trends.

Following are agents which are formalized in our agent base model:

Noise Traders make uninformed decisions influenced by market noise and sentiment, typically not optimizing for slippage and fees, and have a medium to high stickiness due to platform familiarity or incentives. Their market influence is low, but they represent a significant portion of the user base.

MEV Bots employ complex strategies to exploit miner-extractable value, with a focus on transactions like front-running. They have low loyalty to any platform but play a significant role in trading volume due to their high-influence strategies.

Whale Traders aim to influence the market minimally while executing large orders. They use sophisticated strategies like TWAP and iceberg orders to minimize market impact and are not particularly loyal to a single exchange.

Arbitrageurs are always on the lookout for price discrepancies between pools or DEXes. They engage in trades only when there is a profitable opportunity, after accounting for gas fees and slippage.

Static LPs provide liquidity based on simple criteria like price and time, showing high stickiness due to slower responses to market dynamics and a tendency to support pools aligned with their long-term views.

Grid Strategy LPs use a methodical approach to liquidity provision, setting multiple levels based on market prices and adjusting their positions dynamically to respond to price changes.

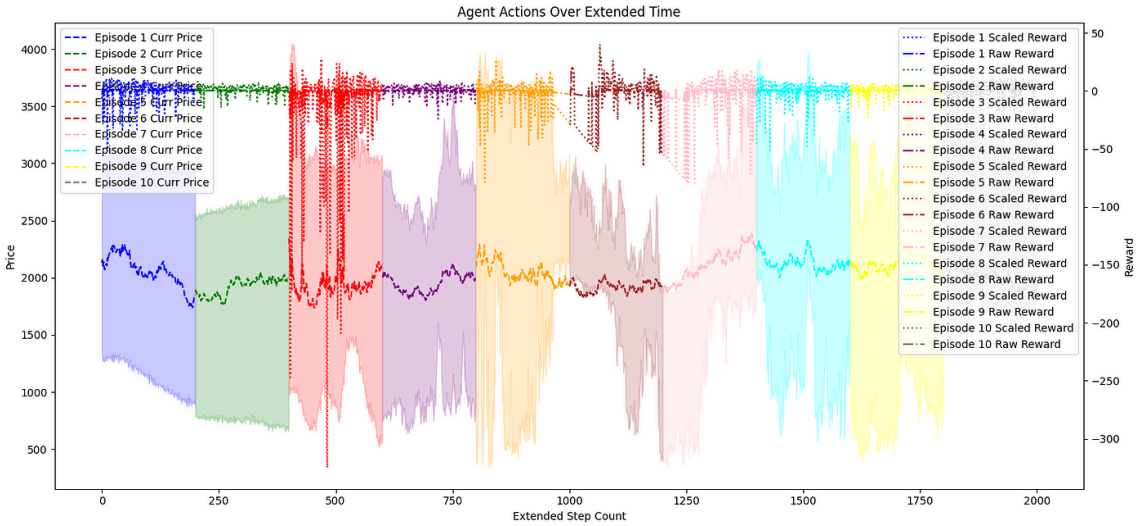
Dynamic Strategy LPs actively adapt to market conditions, assessing the Return on Liquidity to dynamically adjust their portfolio and optimize the allocation of their assets.

Training

Rl agents (DDPG and PPO) are trained for 1000 steps while changing the underlying pool in each episode. We used a different pool in each episode to make our environment more dynamic providing agents with different types of pools (low/high liquidity pool, stable/volatile pool, different fee tiers etc.). Following are some results of training our agents

Agent Actions Over Extended Time

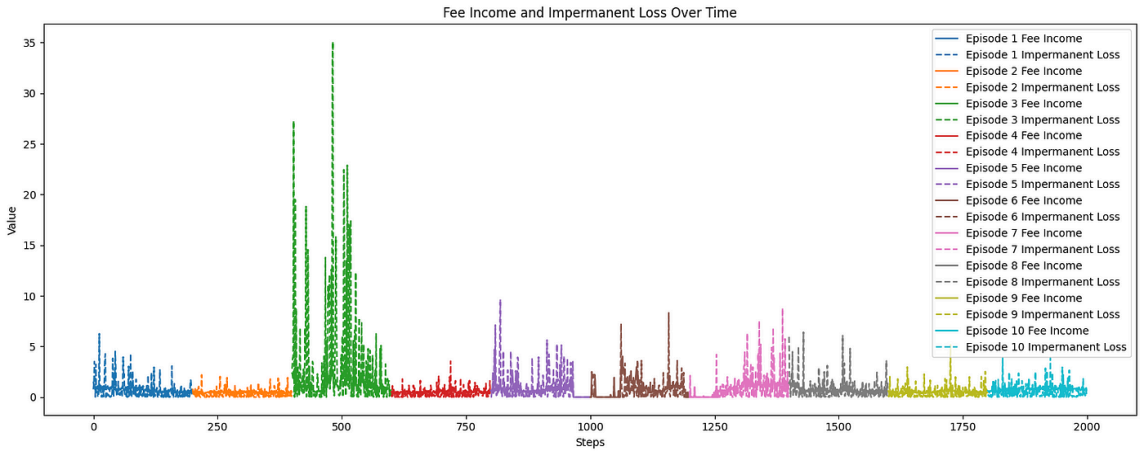
Figure 1: Market Price and Reward Dynamics



This figure juxtaposes the agent's actions with market prices across ten training episodes. The colored lines represent the market prices, while the shaded areas show the agent's rewards. The plot indicates how the agent's decisions align with market movements and the resulting rewards, which are a function of fees earned minus impermanent loss.

Fee Income and Impermanent Loss Over Time

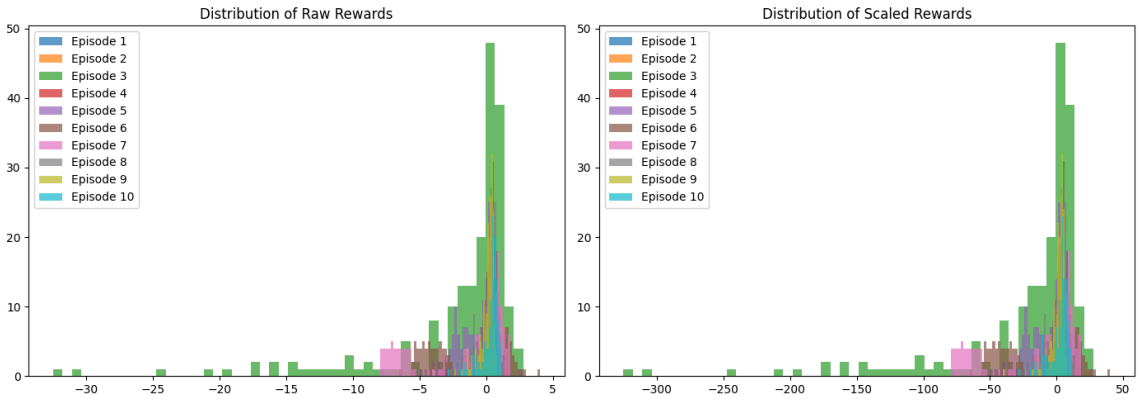
Figure 3: Economic Performance Metrics



This figure tracks the fee income and impermanent loss for each episode over time. Solid lines show the fees the agent earns, and dashed lines represent impermanent loss. Episodes where fee income is higher than impermanent loss indicate profitable strategies, showing the agent's ability to optimize its liquidity provision.

Distribution of Raw and Scaled Rewards

Figure 2: Reward Frequency Analysis



The histograms provide a frequency analysis of the rewards that the agent receives. The left plot shows the distribution of raw rewards, reflecting immediate profitability, while the right plot adjusts these rewards to a common scale to aid in the agent's learning process. The distributions help evaluate the consistency of the agent's performance.

Evaluation

To evaluate the results of trained RL Agents (PPO,DDPG), we performed a comparative analysis of trained agent's performance against baseline agent's performance across the performance

metrics rewards (raw and scaled), fee income and impermanent loss

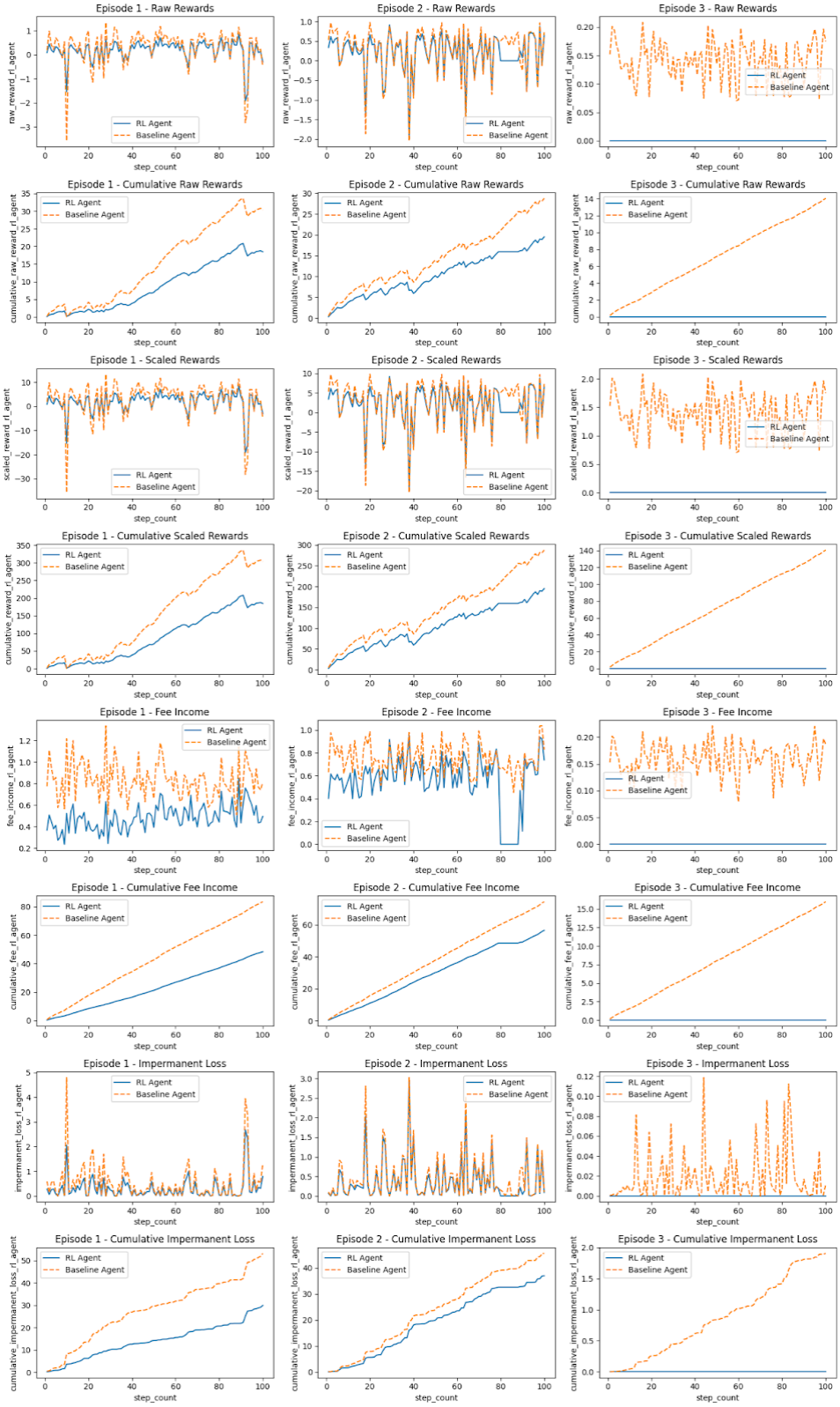
Baseline Strategy

The baseline strategy utilized to evaluate the performance of RL Agents is Maverick's mode both. Maverick's Mode Both, is an advanced liquidity provision strategy that dynamically adjusts an LP's position in response to market price movements in either direction — right or left — aiming to optimize fee income while managing risks associated with impermanent loss and asset reallocation.

Evaluation Results

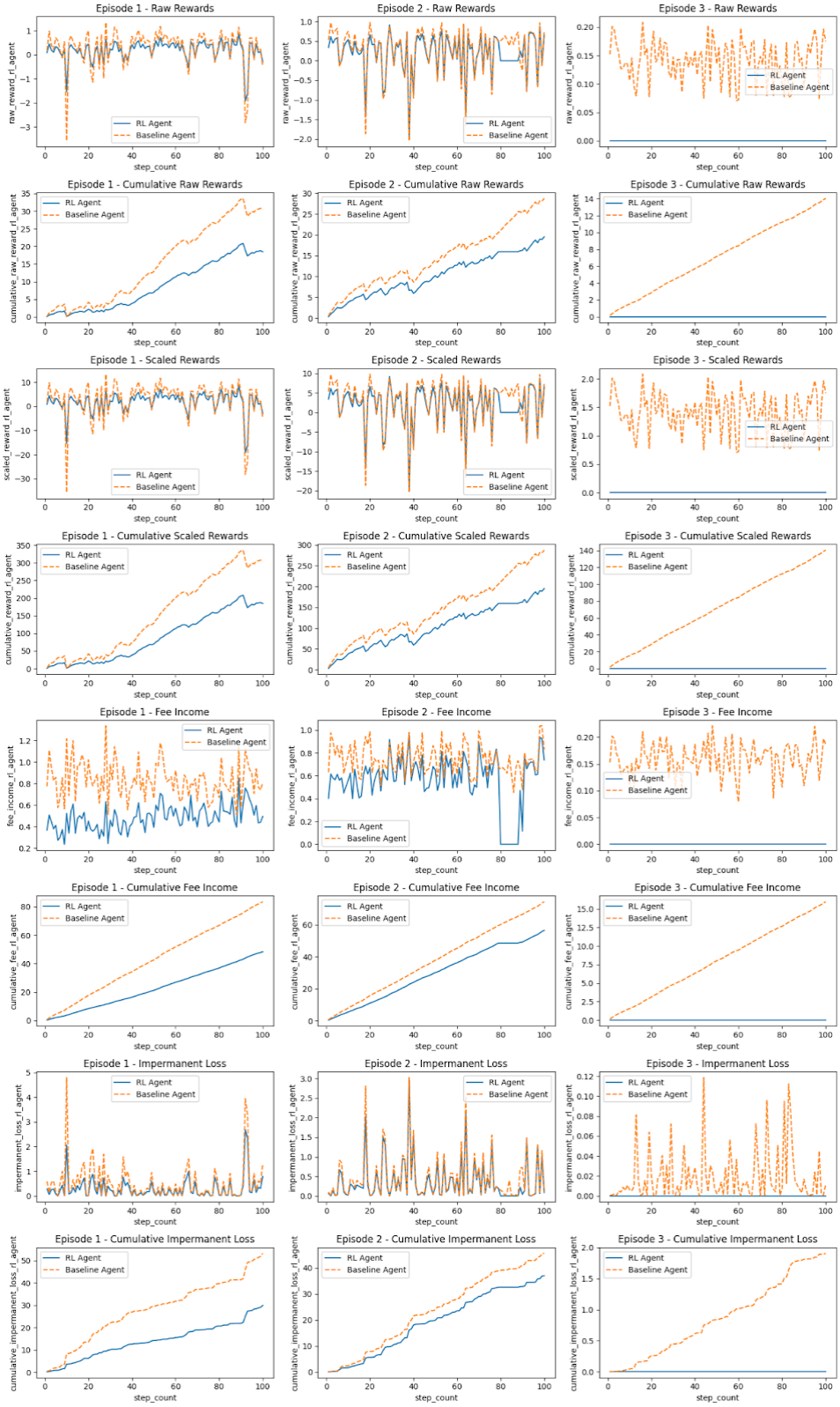
Our evaluation of reinforcement learning (RL) agents against a Mode Both-inspired baseline strategy yields a set of comprehensive metrics.

Raw Rewards:


Episode 1 - Raw Rewards
Episode 2 - Raw Rewards
Episode 3 - Raw Rewards
Episode 1 - Cumulative Raw Rewards
Episode 2 - Cumulative Raw Rewards
Episode 3 - Cumulative Raw Rewards
Episode 1 - Scaled Rewards
Episode 2 - Scaled Rewards
Episode 3 - Scaled Rewards
Episode 1 - Cumulative Scaled Rewards
Episode 2 - Cumulative Scaled Rewards
Episode 3 - Cumulative Scaled Rewards
Episode 1 - Fee Income
Episode 2 - Fee Income
Episode 3 - Fee Income
Episode 1 - Cumulative Fee Income
Episode 2 - Cumulative Fee Income
Episode 3 - Cumulative Fee Income
Episode 1 - Impermanent Loss
Episode 2 - Impermanent Loss
Episode 3 - Impermanent Loss
Episode 1 - Cumulative Impermanent Loss
Episode 2 - Cumulative Impermanent Loss
Episode 3 - Cumulative Impermanent Loss
RL Agent
Baseline Agent
step_count
raw_reward_rl_agent
cumulative_raw_reward_rl_agent
scaled_reward_rl_agent
cumulative_reward_rl_agent
fee_income_rl_agent
cumulative_fee_rl_agent
impermanent_loss_rl_agent
cumulative_impermanent_loss_rl_agent

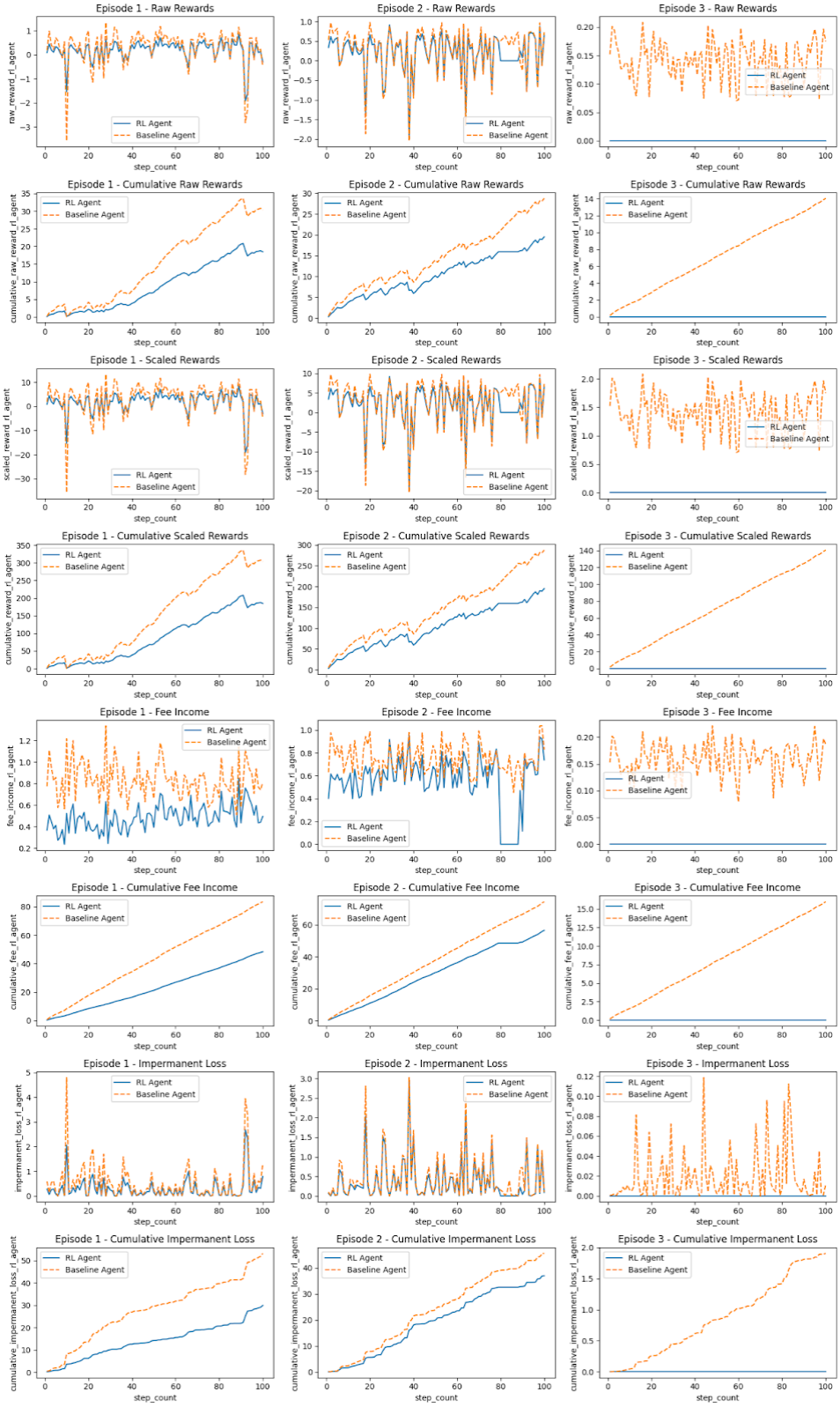
An immediate observation from the raw rewards plots is the volatility in the RL agents' performance, with rewards oscillating sharply compared to the baseline. This suggests that the RL agents are responding to market conditions with a higher frequency of position adjustments, which could be indicative of a more aggressive or responsive strategy. However, when we examine the cumulative raw rewards, we observe that the RL agents consistently accumulate higher rewards over time across all episodes, pointing to the effectiveness of their dynamic decision-making process despite the increased action volatility.

Scaled Rewards:


Episode 1 - Raw Rewards
Episode 2 - Raw Rewards
Episode 3 - Raw Rewards
Episode 1 - Cumulative Raw Rewards
Episode 2 - Cumulative Raw Rewards
Episode 3 - Cumulative Raw Rewards
Episode 1 - Scaled Rewards
Episode 2 - Scaled Rewards
Episode 3 - Scaled Rewards
Episode 1 - Cumulative Scaled Rewards
Episode 2 - Cumulative Scaled Rewards
Episode 3 - Cumulative Scaled Rewards
Episode 1 - Fee Income
Episode 2 - Fee Income
Episode 3 - Fee Income
Episode 1 - Cumulative Fee Income
Episode 2 - Cumulative Fee Income
Episode 3 - Cumulative Fee Income
Episode 1 - Impermanent Loss
Episode 2 - Impermanent Loss
Episode 3 - Impermanent Loss
Episode 1 - Cumulative Impermanent Loss
Episode 2 - Cumulative Impermanent Loss
Episode 3 - Cumulative Impermanent Loss
RL Agent
Baseline Agent
step_count
raw_reward_rl_agent
cumulative_raw_reward_rl_agent
scaled_reward_rl_agent
cumulative_reward_rl_agent
fee_income_rl_agent
cumulative_fee_rl_agent
impermanent_loss_rl_agent
cumulative_impermanent_loss_rl_agent

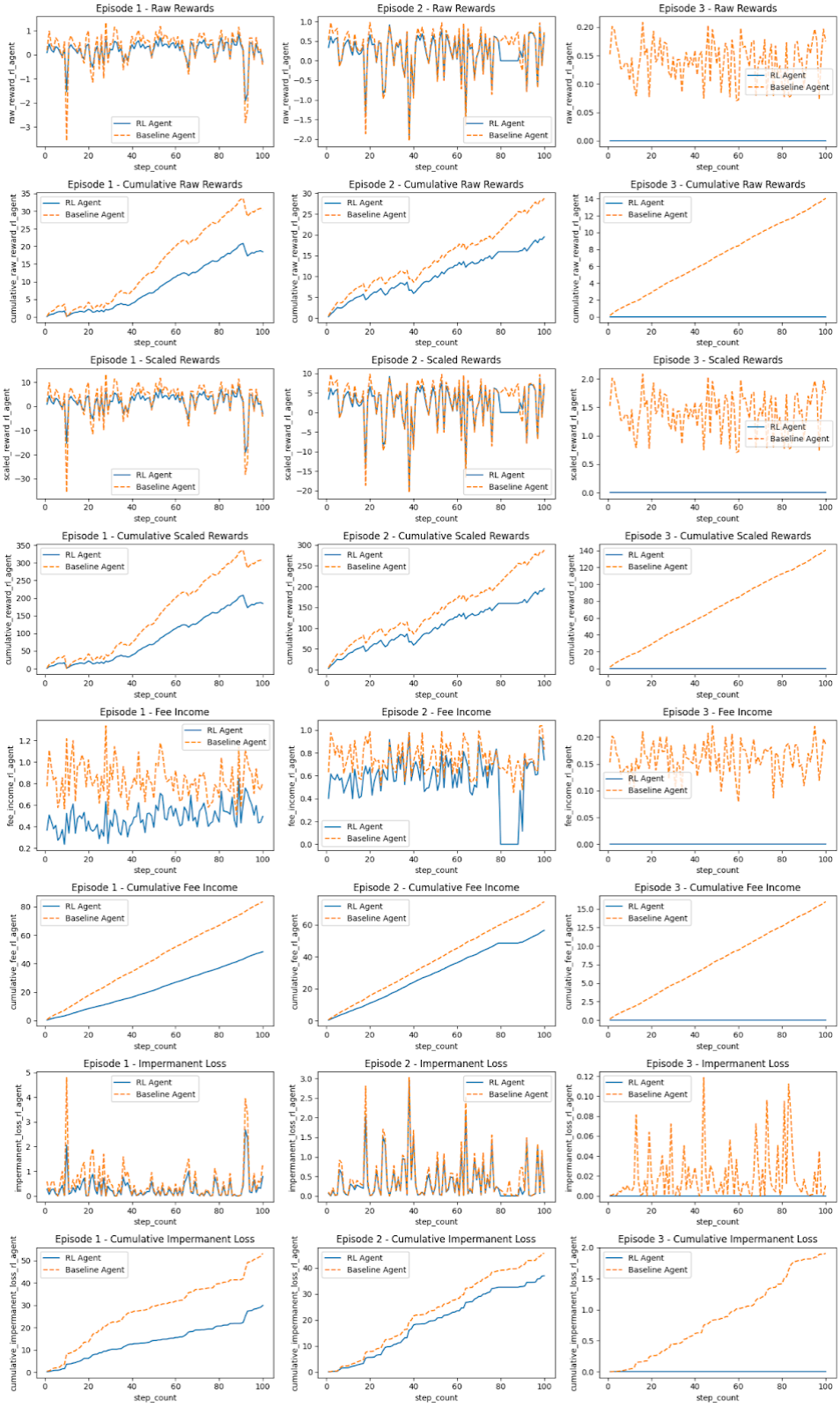
The scaled rewards charts further elucidate the performance of the RL agents, where the magnitude of fluctuations is significantly reduced. This suggests a normalization process that aids in stabilizing the learning and evaluation procedure. Cumulative scaled rewards plots reinforce the superiority of the RL agents over the baseline, with a steady and more pronounced divergence from the baseline's performance. This indicates that the RL agents are not only more active but also more effective in optimizing their reward function, likely balancing fee income and impermanent loss adeptly.

Fee Income:


Episode 1 - Raw Rewards
Episode 2 - Raw Rewards
Episode 3 - Raw Rewards
Episode 1 - Cumulative Raw Rewards
Episode 2 - Cumulative Raw Rewards
Episode 3 - Cumulative Raw Rewards
Episode 1 - Scaled Rewards
Episode 2 - Scaled Rewards
Episode 3 - Scaled Rewards
Episode 1 - Cumulative Scaled Rewards
Episode 2 - Cumulative Scaled Rewards
Episode 3 - Cumulative Scaled Rewards
Episode 1 - Fee Income
Episode 2 - Fee Income
Episode 3 - Fee Income
Episode 1 - Cumulative Fee Income
Episode 2 - Cumulative Fee Income
Episode 3 - Cumulative Fee Income
Episode 1 - Impermanent Loss
Episode 2 - Impermanent Loss
Episode 3 - Impermanent Loss
Episode 1 - Cumulative Impermanent Loss
Episode 2 - Cumulative Impermanent Loss
Episode 3 - Cumulative Impermanent Loss
RL Agent
Baseline Agent
step_count
raw_reward_rl_agent
cumulative_raw_reward_rl_agent
scaled_reward_rl_agent
cumulative_reward_rl_agent
fee_income_rl_agent
cumulative_fee_rl_agent
impermanent_loss_rl_agent
cumulative_impermanent_loss_rl_agent

The fee income plots present a nuanced picture. The RL agents demonstrate a capacity for high peaks in fee income, suggesting moments of optimal liquidity placement. However, these peaks are matched with troughs, implying periods of suboptimal positioning or market downturns. The cumulative fee income plots, on the other hand, reveal a clear trend of outperformance by the RL agents, underscoring their ability to generate higher fee income over time compared to the baseline strategy.

Impermanent Loss:


Episode 1 - Raw Rewards
Episode 2 - Raw Rewards
Episode 3 - Raw Rewards
Episode 1 - Cumulative Raw Rewards
Episode 2 - Cumulative Raw Rewards
Episode 3 - Cumulative Raw Rewards
Episode 1 - Scaled Rewards
Episode 2 - Scaled Rewards
Episode 3 - Scaled Rewards
Episode 1 - Cumulative Scaled Rewards
Episode 2 - Cumulative Scaled Rewards
Episode 3 - Cumulative Scaled Rewards
Episode 1 - Fee Income
Episode 2 - Fee Income
Episode 3 - Fee Income
Episode 1 - Cumulative Fee Income
Episode 2 - Cumulative Fee Income
Episode 3 - Cumulative Fee Income
Episode 1 - Impermanent Loss
Episode 2 - Impermanent Loss
Episode 3 - Impermanent Loss
Episode 1 - Cumulative Impermanent Loss
Episode 2 - Cumulative Impermanent Loss
Episode 3 - Cumulative Impermanent Loss
RL Agent
Baseline Agent
step_count
raw_reward_rl_agent
cumulative_raw_reward_rl_agent
scaled_reward_rl_agent
cumulative_reward_rl_agent
fee_income_rl_agent
cumulative_fee_rl_agent
impermanent_loss_rl_agent
cumulative_impermanent_loss_rl_agent

Impermanent loss graphs present the most striking contrast between the RL agents and the baseline strategy. The baseline maintains a near-zero impermanent loss throughout, while the RL agents exhibit sharp spikes. Despite this, the cumulative impermanent loss for RL agents remains competitive, suggesting that the moments of higher impermanent loss are effectively offset by periods of strategic positioning that minimize such losses or by accruing higher fees.

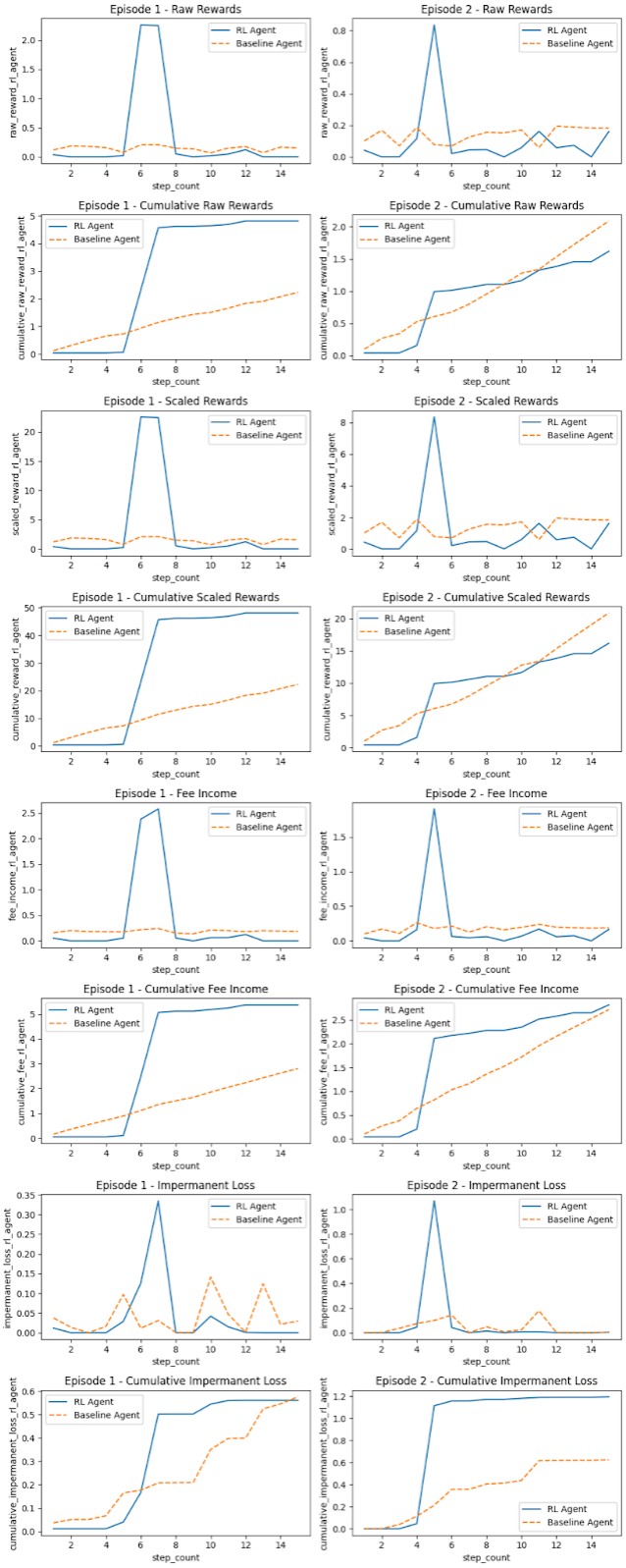
Overall Analysis

The evaluation results suggest that the RL agents, when pitted against the Mode Both-inspired baseline strategy, are capable of adapting their liquidity provisioning strategies to effectively navigate the stochastic nature of DeFi markets. The RL agents' higher cumulative rewards and fee income, alongside

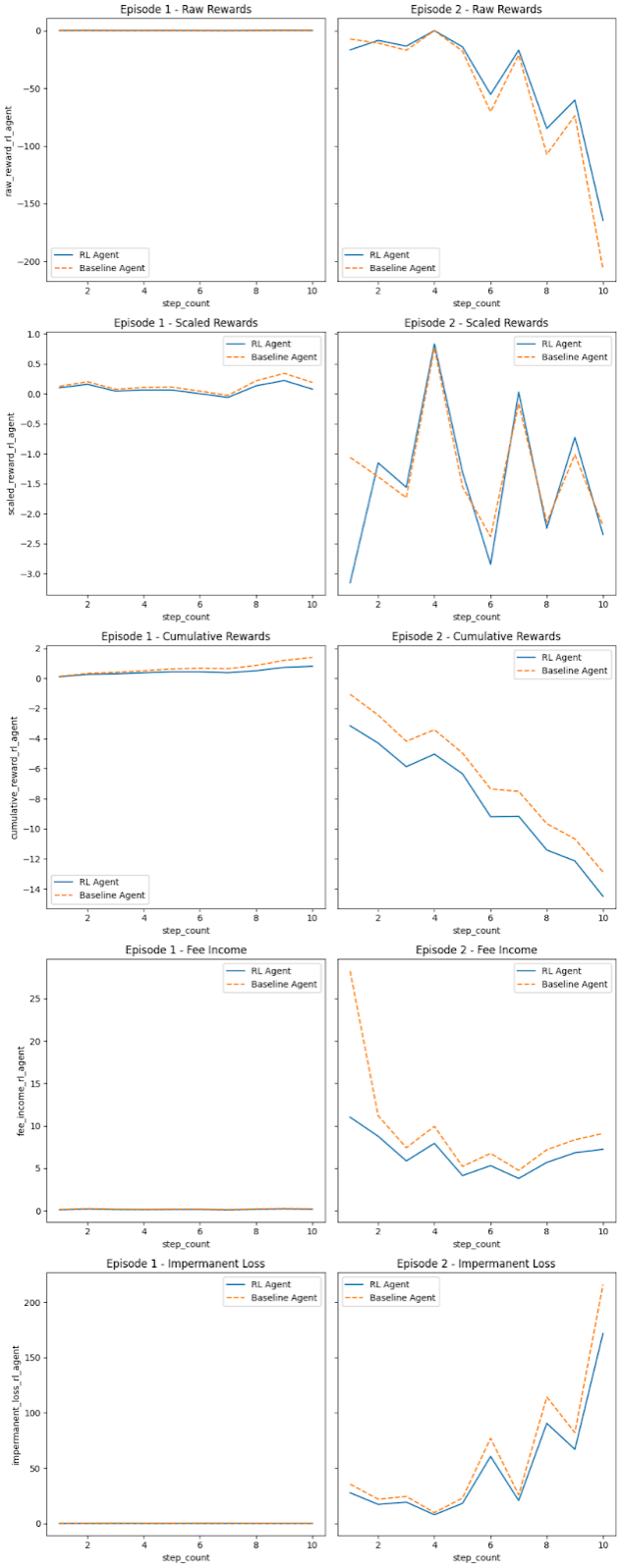
manageable impermanent loss, indicate a sophisticated balancing act between risk and return.

While the baseline strategy provides a consistent and less volatile performance, it is outpaced by the RL agents' capacity to capitalize on market movements and generate higher fees. The trade-off appears to be a greater exposure to impermanent loss, which the RL agents seem to mitigate over time through strategic adjustments.

DDPG Eval


Episode 1 - Raw Rewards
Episode 2 - Raw Rewards
Episode 1 - Cumulative Raw Rewards
Episode 2 - Cumulative Raw Rewards
Episode 1 - Scaled Rewards
Episode 2 - Scaled Rewards
Episode 1 - Cumulative Scaled Rewards
Episode 2 - Cumulative Scaled Rewards
Episode 1 - Fee Income
Episode 2 - Fee Income
Episode 1 - Cumulative Fee Income
Episode 2 - Cumulative Fee Income
Episode 1 - Impermanent Loss
Episode 2 - Impermanent Loss
Episode 1 - Cumulative Impermanent Loss
Episode 2 - Cumulative Impermanent Loss
RL Agent
Baseline Agent
step_count
raw_reward_rl_agent
cumulative_raw_reward_rl_agent
scaled_reward_rl_agent
cumulative_reward_rl_agent
fee_income_rl_agent
cumulative_fee_rl_agent
impermanent_loss_rl_agent
cumulative_impermanent_loss_rl_agent

PPO Eval


Episode 1 - Raw Rewards
Episode 2 - Raw Rewards
raw_reward_rl_agent
Episode 1 - Scaled Rewards
Episode 2 - Scaled Rewards
scaled_reward_rl_agent
Episode 1 - Cumulative Rewards
Episode 2 - Cumulative Rewards
cumulative_reward_rl_agent
Episode 1 - Fee Income
Episode 2 - Fee Income
fee_income_rl_agent
Episode 1 - Impermanent Loss
Episode 2 - Impermanent Loss
impermanent_loss_rl_agent
step_count
RL Agent
Baseline Agent

GitHub

References

https://medium.com/blockapex/intelligent-liquidity-provisioning-framework-d75515381a67

https://medium.com/blockapex/market-making-mechanics-and-strategies-4daf2122121c

https://www.gauntlet.xyz/resources/uniswap-user-cohort-analysis

https://gov.uniswap.org/t/uniswap-incentive-design-analysis/21662

https://arxiv.org/pdf/2108.07806.pdf

https://www.researchgate.net/publication/341848292_Market_makers_activity_behavioural_and_agent_based_approach

https://fruct.org/publications/volume-29/fruct29/files/Struc.pdf

https://www.arxiv-vanity.com/papers/1911.03380/

https://kth.diva-portal.org/smash/get/diva2:1695877/FULLTEXT01.pdf

https://arxiv.org/pdf/2305.15821.pdf

https://github.com/KodAgge/Reinforcement-Learning-for-Market-Making/tree/main

https://arxiv.org/ftp/arxiv/papers/2211/2211.01346.pdf

https://arxiv.org/pdf/2004.06985.pdf

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=968268

z

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0277042

https://deliverypdf.ssrn.com/delivery.php?ID=104119098102026014120072084014107007042068069003049020126088025087121115103007084028042013055035009000054122074096068089064070102052026003014069082076098016080066026088066039027093020006122067093104092065070020126069068106118079127088008098077106031120&EXT=pdf&INDEX=TRUE

Intelligent Liquidity Provisioning Framework V2 — Implementation

The Intelligent Liquidity Provisioning (ILP) Framework V2 for Uniswap V3 pools is an advanced system that leverages agent-based modeling (ABM) and reinforcement learning (RL) to optimize liquidity provisioning strategies. Building on the success of V1, this version introduces a suite of enhancements aimed at providing more accurate simulations, improved decision-making capabilities, and an efficient inference process. Additionally, V2 features a Django app for easy interaction with the framework through API endpoints.

V2 Framework GitHub:

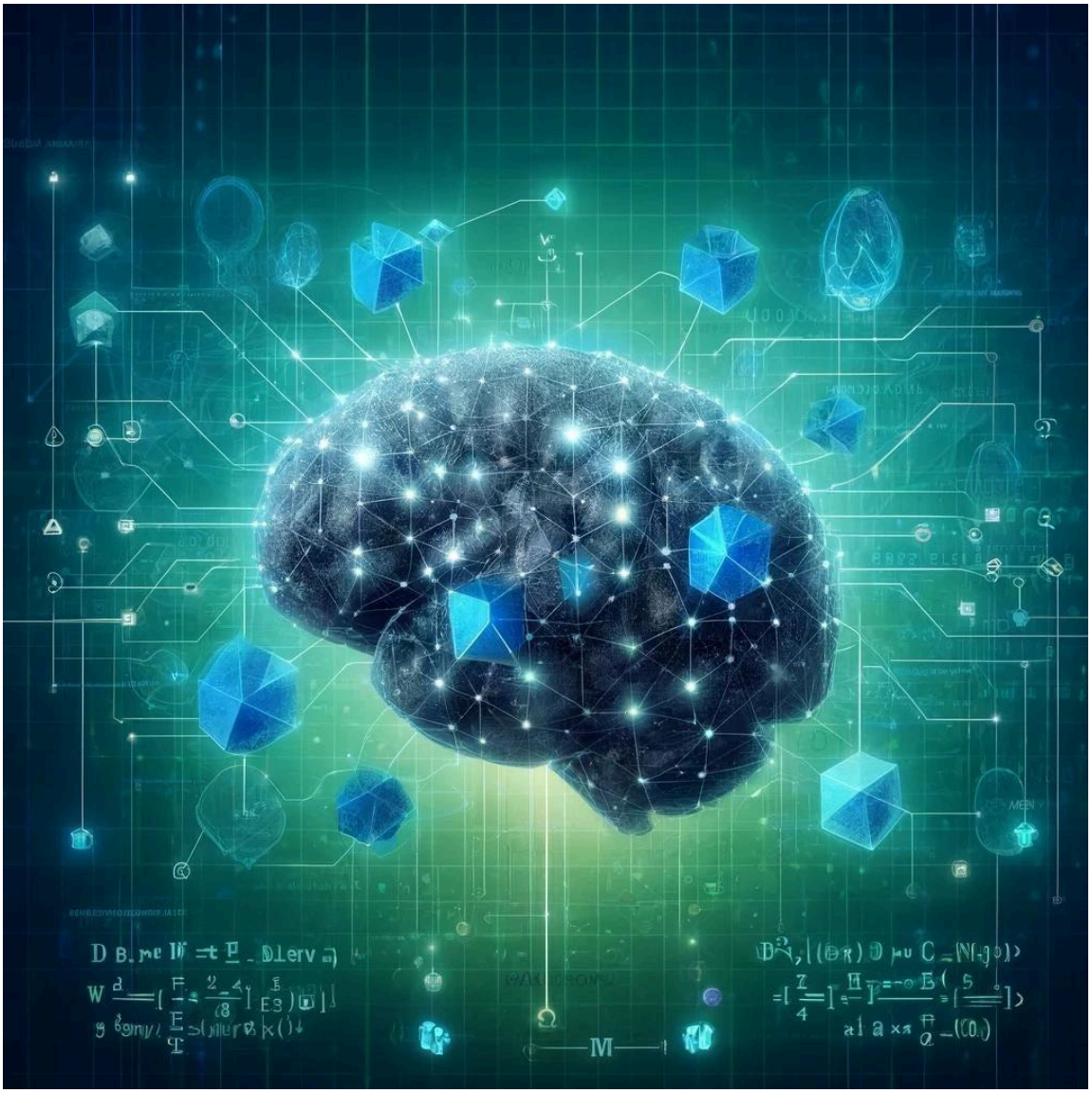
https://github.com/idrees535/Intelligent-Liquidity-Provisioning-Framework-V2

V1 Framework GitHub:

https://github.com/idrees535/Intelligent-Liquidity-Provisioning-Framework-V1

Article:

https://medium.com/blockapex/intelligent-liquidity-provisioning-framework-d75515381a67



Key Improvements Over V1

1. **Integration of PPO Agent:** Proximal Policy Optimization (PPO) agent has been added to our RL algorithms, offering

better stability and performance in training over the DDPG agent.

2. **Enhanced Uniswap V3 Model:** The Uniswap V3 model has been improved to better handle edge cases in swaps and liquidity provisioning.
3. **Max Budget Utilization:** A new feature to ensure liquidity is provided within the user-defined maximum budget.
4. **Enhanced Evaluation Strategy:** More robust evaluation metrics and strategies have been implemented to assess the performance of liquidity provisioning strategies more accurately.
5. **Improved Training and Evaluation Visualizations:** Training and evaluation processes now feature more comprehensive metrics for a clearer understanding of model performance.
6. **Custom Reward Functions:** Introduction of custom reward functions based on liquidity providers' (LPs)

preferences, allowing for more personalized strategy optimization.

7. **Improved Inference Process:** Enhanced inference process to better incorporate LPs' risk aversion and other preferences into decision-making.
8. **Django Application for API Endpoints:** A new Django application has been developed to expose model functionalities via API endpoints for easier integration and automation.
9. **Integration with MLflow for Experiment Tracking:** Incorporates MLflow for tracking experiments, ensuring a systematic way to log, visualize, and compare model training sessions.

Usage

This framework offers a set of API endpoints to initialize the environment, train and evaluate models using both DDPG and PPO

algorithms, and perform inference based on the Uniswap V3 model and user-defined parameters. Below are example `curl` commands to interact with these endpoints.

Initialize Script

To set the base path and reset environment variables for the framework, use the following command:

```
curl -X POST http://127.0.0.1:8000/initialize_script/ \

-H "Content-Type: application/json" \

-d '{

   "base_path": "<path_to_your_cloned_repository>",

   "reset_env_var": true

}'
```

Replace `<path_to_your_cloned_repository>` with the actual path where the repository is located on your system.

Train DDPG Agent

To train the DDPG agent with custom parameters:

```
curl -X POST http://127.0.0.1:8000/train_ddpg/ \

-H "Content-Type: application/json" \

-d '{

  "max_steps": 2,

  "n_episodes": 2,

  "model_name": "model_storage/ddpg/ddpg_fazool",

  "alpha": 0.001,

  "beta": 0.001,
```

```
    "tau": 0.8,

    "batch_size": 50,

    "training": true,

    "agent_budget_usd": 10000,

    "use_running_statistics": false

}'
```

Evaluate DDPG Agent

To evaluate the DDPG agent:

```
curl -X POST http://127.0.0.1:8000/evaluate_ddpg/ \

-H "Content-Type: application/json" \

-d '{

    "eval_steps": 2,
```

```
    "eval_episodes": 2,

    "model_name": "model_storage/ddpg/ddpg_fazool",

    "percentage_range": 0.6,

    "agent_budget_usd": 10000,

    "use_running_statistics": false

}'
```

Train PPO Agent

To train the PPO agent with custom parameters:

```
curl -X POST http://127.0.0.1:8000/train_ppo/ \

-H "Content-Type: application/json" \

-d '{

    "max_steps": 2,
```

```
    "n_episodes": 2,

    "model_name": "model_storage/ppo/ppo2_fazool22",

    "buffer_size": 5,

    "n_epochs": 20,

    "gamma": 0.5,

    "alpha": 0.001,

    "gae_lambda": 0.75,

    "policy_clip": 0.6,

    "max_grad_norm": 0.6,

    "agent_budget_usd": 10000,

    "use_running_statistics": false,

    "action_transform": "linear"
```

```
}'
```

Evaluate PPO Agent

To evaluate the PPO agent:

```
curl -X POST http://127.0.0.1:8000/evaluate_ppo/ \

-H "Content-Type: application/json" \

-d '{

  "eval_steps": 2,

  "eval_episodes": 2,

  "model_name": "model_storage/ppo/ppo2_fazool",

  "percentage_range": 0.5,

  "agent_budget_usd": 10000,

  "use_running_statistics": false,
```

```
    "action_transform": "linear"

}'
```

Perform Inference

To perform inference using the model, providing details about the pool state and user preferences:

```
curl -X POST http://127.0.0.1:8000/inference/ \

-H "Content-Type: application/json" \

-d '{

    "pool_state": {

        "current_profit": 500,

        "price_out_of_range": false,

        "time_since_last_adjustment": 40000,
```

```
        "pool_volatility": 0.2

    },

    "user_preferences": {

        "risk_tolerance": {"profit_taking": 50, "stop_loss": -500},

        "investment_horizon": 7,

        "liquidity_preference": {"adjust_on_price_out_of_range":
true},

        "risk_aversion_threshold": 0.1,

        "user_status": "new_user"

    },

    "pool_id": "0x4e68ccd3e89f51c3074ca5072bbac773960dfa36",

    "ddpg_agent_path": "model_storage/ddpg/ddpg_1",
```

```
    "ppo_agent_path":
"model_storage/ppo/lstm_actor_critic_batch_norm"


}
```

These commands serve as a starting point for interacting with the framework. Customize the JSON payload as needed to fit your specific requirements for training, evaluation, and inference.

ILP Agent Framework V3

ILP agent framewrok let's liquidty providers/liquidty managers to build/devlop, train and deploy their liquidity managememnt agents with their custom insticts and prefernces. An ILP Agent consists if three differnt type of agents working together to efficiently manage liquidty On Uniswap V3. Following are three differnt components of this framewrok:

1. **Strategy Agent**

This agent mamnges the budget of vault/portofolio. Basically it allocates budget across differnt lqiuidty pools, and manges the distribution of budget based on historical performmance data of the lqiuidty pools, while incorporating the prefernces of user (Retail LP/ LM). User prefernce include total budget, investment horizon, risk factor, stop loss etc. Based on this stratgy agent allocates the total budget in differnt pools and speciifices the amount of budget allocated to each pool, and in case of multiple position based stratgy it alos speciifies the amount of budget allocated to each position in pool. The output of strategy agent for some input i.e {10000USD,30 Dasys, 0.6 Risk factor} willl be position 1: ETH/USD Pool, 3000USD, position 2 ETH/USD Pool, 1000USD, position 3: BTC/WETH Pool, 5000USD

This strategu agent also monitors these liquidty position on epoch basis i.e daily and then based on the progress and goals of the overall

startegy adjusts these positions, it can remove positions, rebalance positions or add new posistions for ecrtain pools

2. **Predictor Agent**

This agent based on the output of stratgy agent {predicts the tick_lower,tick_upper} of each add position action suggested by stratgy agent. This agent is basically the current RL ILP agnet in V2 framework which given the state space predicts a liquidty position in given pool

3. **Executor Agent**

This executor agent executes the actions of stratgy agent and predictor agent. This is where giza agents platform which provides a secure and verifiable infrastructure of deploytemnt of these agents in production

Implementtaion

Improvements in Current RL Agent

1. Integarte voyager simulator

2. Improve policy functions

3. Improve obs space, action space and reward function

Impelent Stratgy Agent

1. A gen AI model which is fine tuned on liquidty provisioning, portofolio managemnt and risk managemnt data, which devlops optimal budget allocation startgeies across pools

2. Function calling/tool calling capabilities to analyze the histroical data

3. Functions yto traanslate the user prefernces/risk appetite in stratgy actions

4. Monitor current LP positions and quantiatvely measure their performance across benchmarks and baselines

5. Based on monitoring analysis dynamicallya djust stratgies

Integrate these agents with giza agents

1. Automate contarct calls execute stratgy actions

2. Manage funds through smart account in secure environemnt

3. Authrize transaction execution in an autonoumous using permissions assigend in smart account