QLAMMP: A Q-Learning Agent for Optimizing Fees on Automated Market Making Protocols

Dev Churiwala  
Viterbi School of Engineering*  
University of Southern California  
Los Angeles, USA  
f20180602@goa.bits-pilani.ac.in

Bhaskar Krishnamachari  
Viterbi School of Engineering  
University of Southern California  
Los Angeles, USA  
bkrishna@usc.edu

Abstract—Automated Market Makers (AMMs) have cemented themselves as an integral part of the decentralized finance (DeFi) ecosystem. AMMs are a type of exchange that allows users to trade assets without the need for a centralized exchange. They form the foundation for numerous decentralized exchanges (DEXs), which help facilitate the quick and efficient exchange of on-chain tokens. All present-day popular DEXs are static protocols, with fixed parameters controlling the fee and the curvature - they suffer from invariance and cannot adapt to quickly changing market conditions. This characteristic may cause traders to stay away during high slippage conditions brought about by intractable market movements. We propose a Reinforcement Learning (RL) framework to optimize the fees collected on an AMM protocol. In particular, we develop a Q-Learning Agent for Market Making Protocols (QLAMMP) that learns the optimal fee rates and leverage coefficients for a given AMM protocol and maximizes the expected fee collected under a range of different market conditions. We show that QLAMMP is consistently able to outperform its static counterparts under all the simulated test conditions.

Index Terms—automated market maker, blockchain, decentralized exchange, decentralized finance, reinforcement learning

I. INTRODUCTION

With the ever-growing interest in and adoption of blockchain and cryptocurrency technology among common actors, institutions, and academicians, Decentralized Finance (DeFi) applications have rapidly gained popularity. DeFi's rise preserves the general principles of scalability, security, and decentralization. One of its core advancements was the adoption of Decentralized Exchanges (DEXs) built atop the Automated Market Maker (AMM) model. This revolutionized DeFi, bringing efficient trading with no restrictions, complete anonymity, and quick settlement to the table. Academicians have recently begun exploring these DEXs and released Systematization of Knowledge works detailing the protocols’ functioning [1]–[3]. We concern ourselves with these DEXs, which have also independently accounted for a staggering $1.5 trillion in aggregated transaction volumes over the past year [4].

Traditionally, most AMM protocols have had static parameters controlling their fee rate and curvature - together controlling the protocol’s bid/ask prices, which indirectly govern the swap volume and, in turn, the total accrued fees. Let us consider the direct implications of such a model. Let's say the protocol is optimized to perform best when users swap large amounts with higher tolerances to loss. Now, consider the user demands changing (perhaps due to external market factors) in terms of their traded volume or slippage appetite, the protocol retains its previous invariant configuration and likely performs poorly. This paper aims to solve this dynamic user problem by adapting the fee rate and curve behavior based on market conditions. For this purpose, we propose an RL agent that modifies the protocol parameters at regular intervals, referred to as Q-learning Agent for Market Making Protocols (QLAMMP).

RL is a well-established technique that has the ability to learn and build knowledge about dynamic environments [5]. We propose the use of the Q-learning RL algorithm to train our agent [6]. By interacting with the environment, the proposed agent builds a knowledge base of the environment, actions, and expected rewards given the constantly changing conditions. In this way, the agent can converge on an optimal policy that maximizes the expected total rewards, in this case, the fees accrued by the protocol. Hence, Q-learning is a promising approach to learning policies in an uncontrolled environment, such as the ever-shifting demands of a DEX’s user base. To our knowledge, this is the first time RL has been used to optimize fees on an AMM protocol.

The main contributions of this paper are as follows:
• We are the first to study the use of an RL agent to optimize the fees on an AMM protocol by modifying fee rates and the curve behavior.
• We hypothesize and evaluate the agent’s performance under a slew of different conditions.
• We empirically evaluate the effect that the size of the updating interval (measured as an update made every k number of steps) has on the agent’s performance.
• We also look at the standard deviation of the agent's actions through the training epochs to ensure that the...
agent is adapting to the environment and taking different actions rather than just optimizing for and sticking to a condition-agnostic state.

- To interpret QLAMMP’s performance, we also break down the agent’s behavior into action space components - a subset of the actions modifying just the fee rate, another subset of actions modifying just the curvature - thus seeing the effect that a particular subset of the actions has on the agent’s performance.

- To accommodate for the dynamic user conditions and achieve robust agent performance under varied conditions - we develop a modular training environment, which can be adapted to fit numerous different AMM protocols - thus allowing for the agent’s training based on specific use cases.

- We also touch upon the practical aspect of maintaining decentralization and privacy using a decentralized oracle or on-chain deployment of QLAMMP.

The remainder of the paper is structured as follows. In Section II, we review the related work. Section III provides a basic background on AMMs needed to parse the rest of the paper. Section IV presents our proposed system model, and Section V describes the associated problem formulation. Section VI explains the simulation methodology used, and Section VII presents our results and findings. Finally, we conclude the paper in Section VIII.

II. RELATED WORKS

AMMs were popularized by Hanson’s Logarithmic Market Scoring Rule (LSMR) for prediction markets [7]. Bonding Curve AMMs, like the ones explored in our work, were introduced by the Bancor Protocol [8]. Later, Buterin proposed a curve-based AMM for a DEX, which was implemented as Uniswap [9], [10], and Egorov developed the Curve Finance Protocol, focused on efficient trades for tokens with equivalent pricing [11].

Academic interest in AMMs led to formalizing their mathematical properties and studying optimal arbitrage strategies [12]. Evans et al. developed a framework for selecting fees for Geometric Mean Market Makers (G3Ms) [11]–[3].

Krishnamachari et al. proposed an innovative solution to eliminate arbitrage opportunities in curve-based AMMs using dynamic curves [13]. Wang and Krishnamachari further formulated a method to find optimal trading policies for dynamic curve AMMs using dynamic programming [14].

Research using RL in blockchain systems has tackled decision-making problems, such as optimizing offloading performance in blockchains [15], maximizing transactional throughput in IoV data [16], and trading cryptocurrencies using historical data [17]. Additionally, RL has been applied to analyze attacks on blockchain incentive mechanisms [18], optimize blockchain configurations for healthcare systems [19], and optimize market making on traditional stock markets [20].

However, none of the past works explored an RL framework for optimizing fees in AMM protocols. In our work, we propose such an RL framework to dynamically adjust the protocol’s parameters and maximize fees collected over time.

III. BACKGROUND ON AMMS

Recent advances in developing DEXs have primarily used AMMs at their cores [10], [11], [21]. AMMs do away with the traditional order book system of market making and can provide liquidity even when market action is limited.

A. Fundamental Properties

1) Swap: A swap refers to an exchange of tokens between a user and a counterparty; here, the AMM protocol.

2) Bonding Curve: Each AMM depends on its underlying bonding curve to generate trading prices algorithmically. This bonding curve is defined by a conservation function that follows an invariant property. To understand this, consider the following function.

\[ xy = k \]

Here, constant \( k \)'s value is determined by the initial liquidity provision (when the system is initialized); using this \( k \) and the new quantity of a token, the reserve quantity of the other token can be solved - thus giving us a buy/sell price.

3) Slippage: Slippage, for AMMs, is defined as the percent difference between the spot price and the realized price of a trade. This difference comes about due to two main factors, (1) the change in the bid/ask spread during trade execution and (2) the relative size of the swap compared to the total liquidity in the system. The slippage for a given protocol is particularly pronounced when the liquidity in the system is low or when a swap amount represents a significant fraction of the liquidity pool.

B. Exchange Functions

1) Constant Sum Market Marker (CSMM): For token reserves \((x, y)\), a CSMM maintains the sum of reserves constant; that is, the exchange function satisfies the following equation.

\[ (R_x - \delta_x) + (R_y + \delta_y) = k \]

2) Constant Product Market Maker (CPMM): For token reserves \((x, y)\), a CPMM maintains the product of reserves constant; that is, the exchange function satisfies the following equation.

\[ (R_x - \delta_x) * (R_y + \delta_y) = k \]

3) Hybrid Function Market Maker (HFMM): Given the same initial conditions, the bonding curve for a HFMM lies somewhere between a CSMM and the CPMM curves based on how it is set up. The curvature of an HFMM can be controlled based on a leverage coefficient \( A \) - which decides the skewness of the curve toward either a CPMM curve or a CSMM curve.

IV. SYSTEM MODEL

In this section, we first provide an overview of the system and discuss the hybrid automated market-making protocol used for our simulations. Next, we present the proposed user dynamics and, finally, the swap dynamics - how a user on the protocol carries out each swap.
A. Overview

Our system (Fig. 1) consists of a central AMM protocol that acts as a counter-party to all swaps. The system also has users who request swaps. The users’ decision to swap is based on modeled demand based on the value they get for their tokens. The swaps are modeled as a queue, with the front being serviced first.

B. Protocol Dynamics

We design our proposed Q-Learning Agent to optimize for a hybrid AMM protocol. The protocol’s underlying exchange function can dynamically shift from a constant-product invariant to a constant-sum invariant. This shift is accomplished by interpolating between the two invariants, controlled by a Leverage Coefficient, $A$. When $A \to 0$, the exchange function emulates a constant-product one (like Uniswap); when $A \to +\infty$, the exchange function is essentially a constant-sum one. This behavior is captured by the following invariant proposed in [11]:

$$ A n^x i + D = A D n^x + \frac{D n^{x+1}}{n^x} \prod x_i \quad (4) $$

Our software implementation of the protocol is based on the implementation by the recent survey on AMMs [2]. We modify their implementation by adding fee mechanics to the Curve class. For each successful swap, the protocol collects a set percent of fee, $R_f$, from the input tokens, $T_i$, to get the final tokens, $T_f$.

$$ F = T_i \cdot \frac{R_f}{100} \quad (5) $$

$$ T_f = T_i - F \quad (6) $$

We define a `getLoss()` function, allowing users to query the protocol for the current loss, or price impact, $\varphi$, resulting from slippage and fees.

$$ \varphi = \frac{T_f - \Delta \sum x_i}{T_f} \cdot 100 \quad (7) $$

C. User Dynamics

We have designed our simulation to closely emulate a real user interacting with a financial exchange, making swapping decisions based on empirical factors such as price and slippage and a subjective component - urgency. This modeling is random for each swap, with each parameter of the swap being sampled from carefully chosen random distributions. Assuming there are $N$ users in the system, we let $U = \{U_1, U_2, ..., U_N\}$, denote the set of users. Each user, $i$ ($i \in U$), has a non-negative balance of the two tokens, $\{x_k, y_k\}$ - initialized to 1,000 each, and a number identifying them, $k$.

The environment generates swaps (mechanics discussed in Section VI-C) and assigns them to random users. Upon receiving such a swap order, the user randomly chooses a token index, either 0 or 1. If the balance of the chosen index is less than a given fraction (set to 20% by default) of the other token, indices are inverted. This mechanism prevents the drying up of either token’s balance. Once the swap parameters are finalized, the user queries the protocol for the expected $\varphi$. The swap goes through if the returned $\varphi$ is less than the urgency-modified tolerance. If not, the swap is canceled based on a 40% chance modified by the urgency factor. If the swap is urgent, it is less likely to get canceled and vice-versa. In addition, if the swap is not canceled, the urgency is increased by 1%, and the “holding” swap status is returned to the environment.
D. Swap Dynamics

A swap object, $E_i$, as described by us, consists of a slippage tolerance value, $s_i$, an urgency value, $u_i$, and a user identification number, $k$ ($k \in \{1, 2, \ldots, N\}$). The environment generates a set number of swaps based on the number of users in the system. The environment samples normal distributions for each swap’s tolerance and urgency value. Each of these swaps is randomly assigned to users in the system and pushed into a queue, ready to be executed. Once serviced, the swap is popped from the queue.

These swap objects also hold an amount, token index, and a new flag for swaps that were returned with a “holding” status, it is canceled set to 15) times and fails with a “the swap is tried a predetermined number of times (by default $k = 10$) places in the swap queue and goes through again after $k$ by the user - the object is then pushed back into the swap queue, ready to be executed. Once serviced, the swap is popped from the queue.

V. Problem Formulation

We frame the problem in a way such that one can infer cause and effect relationships from the simulations. Consequently, we propose three different agents:

1) Agent 1, whose actions only change the fees for the protocol.

2) Agent 2, whose actions can only modify the leverage coefficient for the protocol.

3) QLAMMP, whose actions can modify the fees and the leverage coefficient.

These agents allow us to observe the effects of each control vector individually and the combined effect of both.

A. State Space

At each decision epoch, $t$, the state space for the system can be characterized by $S_t = (U, E_t, L_t, A, R_f)$. The meaning of each variable is as follows.

- $U = \{U_1, U_2, \ldots, U_N\}$, denotes the set of users in the system, each user, $i$ ($i \in U$), has their current non-negative balance of the two tokens, $x_i, y_i$.
- $E_t = \text{current swap}$ denotes the swap at the head of the swap queue, it consists of a slippage tolerance value, $s_i$, an urgency value, $u_i$, and a user identification number, $k$ ($k \in \{1, 2, \ldots, N\}$).
- $L_t = \text{slippage}$ denotes the protocol predicted slippage for the given swap, $E_t$. Since this is a continuous - real value, we discretize it.
- $A = \text{leverage coefficient}$ that controls the bonding curve’s curvature.
- $R_f = \text{fee rate}$ denotes the amount of fee collected by the protocol on each swap.

This state space is shared across the aforementioned agents. Upon getting a swap instruction from the environment, the user generates a token index and an amount, which are used to query the protocol for the expected $\phi$. As the agent chooses actions to adapt the fee rate, $R_f$, and the leverage coefficient, $A$ - it attempts to optimize $\phi$ to allow for maximum fee collection and minimum trade cancellations.

B. Action Space

1) Agent 1: This agent has control over the rate of fees of the protocol. This fee rate is doubly bound with a lower limit of 0.04% and an upper limit of 0.30%. These values are derived from the real-world rates of Curve Finance and Uniswap V2, the largest exchanges of their kind, respectively. This range ensures representative taxation on each transaction.

At each decision epoch, $t$, the agent can either increase the fee rate by 1, decrease the fee rate by 1, or leave it as it was previously. This change is governed by the limits mentioned above.

2) Agent 2: This agent controls the protocol’s leverage coefficient, $A$. This coefficient is doubly bound with a lower limit of 0 and an upper limit of 85. Once again, these limits are derived from the real-world coefficients of Uniswap V2 ($A = 0$) and Curve Finance ($A = 85$). This range ensures representative exchange dynamics over all the transactions.

At each decision epoch, $t$, the agent can either increase $A$ by 2, decrease $A$ by 2, or leave it as it was previously. This change is governed by the limits mentioned above.

3) QLAMMP: This agent controls the protocol’s leverage coefficient, $A$, as well as its rate of fees. These parameters are bound by the same limits as Agents 1 & 2. In this way, we can get representative simulations - which can also be compared against each other.

At each decision epoch, $t$, the agent can make one of nine actions. These are a cross-product of the three actions of Agent 1 and the three actions of Agent 2. The larger action domain gives QLAMMP broader control over the protocol and potentially a better shot at optimizing for fees. Once again, all these changes are governed by the limits mentioned above.

C. Rewards

The reward mechanism is shared across the three agents. We model the system by defining three different reward types that are awarded under different conditions.

1) Swap Success: At a decision epoch, $t$, the user, $U_i$, specified by swap, $E_t$, is able to make the swap with the expected slippage, $L_t$, is less than $U_i$’s slippage tolerance, $s_i$.

The agent is awarded a positive value equal to the fee procured by the swap.

2) Swap Holding: At a decision epoch, $t$, the user, $U_i$, was pushed back into the swap queue, $E$, due to the expected slippage, $L_t$, being beyond $U_i$’s slippage tolerance, $s_i$.

The agent is awarded 0 because the outcome of this swap remains to be decided.

3) Swap Canceled: At a decision epoch, $t$, the user, $U_i$, cancels the swap, $E_t$, either due to random chance or because its number of total queries for the swap crossed 15.

In such a case, the agent is awarded -1.

These three reward cases ensure that the agent is rewarded for enabling swaps, and maximizing the procured fee. At the same time, the agent is highly penalized for swaps being...
canceled. This harsh penalty is because, a canceled swap is a direct loss of fees that could have potentially been earned.

VI. SIMULATION METHODOLOGY

Our simulation environment is written in Python using the OpenAI Gym toolkit [22], [23]. We opt for an object-oriented approach to allow for modularity in agents and environment.

A. Reinforcement Learning Algorithm

We utilize the Q-Learning algorithm to train the agent on the given problem. It is a model-free off-policy RL method where the agent’s goal is to obtain an optimal action-value function $Q(s,a)$ by interacting with the environment. It maintains a state-action table $Q(S,A)$ called a Q-table containing Q-values for every state-action pair. At the start, Q-values are initialized to all zeroes. Q-learning updates the Q-values using the Temporal Difference method [6].

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha (R_{t+1} + \gamma \max_a Q(s_{t+1},a_t) - Q(s_t,a_t))$$

where $\alpha$ is the learning rate and $\gamma \in [0,1]$. $Q(s_t,a_t)$ is the actual Q-value for state-action pair $(s_t,a_t)$. The target Q-value for state-action pair $(s_t,a_t)$ is $\{R_{t+1} + \gamma \max_a Q(s_{t+1},a_t)\}$. The table converges using iterative updates for each state-action pair. To efficiently converge the Q-table, an $\epsilon$-greedy approach is used.

The $\epsilon$-greedy approach: At the start of the training, all Q-values are initialized to zero. This uniformity implies that all actions for a state have the same chance to be selected. So to enable the convergence of the Q-table using iterative updates, the exploration-exploitation trade-off is employed. The explorative update for the Q-value of random state-action pair $(s_t,a_t)$ is carried out by randomly selecting the action. The exploitative update selects a greedy action $(a_t)$, with maximum expected rewards, for the state $(s_t)$ from the Q-table.

So, to converge the Q-table from the initial condition (all Q-values are set to zero), the agent initially prioritizes exploration and later it prioritizes exploitation. It uses probability $\epsilon$, to choose random action, or it chooses an action from the Q-table using probability $1 - \epsilon$. In the beginning, the value of $\epsilon$ is one, and it decays with time, tending to zero as the Q-table converges.

$$\epsilon = \epsilon_{\text{min}} + (\epsilon_{\text{max}} - \epsilon_{\text{min}}) \cdot e^{-\eta \cdot \chi}$$

where $\epsilon_{\text{max}}$ & $\epsilon_{\text{min}}$ are the maximum and minimum values for $\epsilon$, predefined by the environment; and $\eta$ is the factor by which $\epsilon$ is reduced every epoch, $\chi$.

B. Simulation Setup

We consider a dynamic exchange function as the underlying curve to facilitate trades on our AMM. The AMM is initialized with 20,000 units of the two tokens. We populate the system with 20 users and 400 swaps per epoch. Under normal swap conditions, each user is given 1,000 units of each token at the beginning of the epoch and under high-liquidity swap conditions the users are initialized with 18,000 units of each token. Each simulation is run over 3,000 epochs.

The Q-Learning algorithm requires discrete values of the observation. Therefore the continuous–real state value is discretized by placing it into 1 of 500 buckets over slippage values ranging from -20% to 20%. The environment is reset to a random state, along with random fee rate & leverage coefficient values at the beginning of each epoch.

C. Generation of Users

Users are generated as a list of objects, $U$, from the User class. Each user is assigned a balance of 1,000 units of each token when the environment is reset. The users are assigned an identification number – equal to their 0-indexed position in the list. There are two additional member variables that a user object has, namely, slippage tolerance and urgency.

1) Slippage Tolerance: There are two different modes of slippage tolerance generation for the user objects.

- Normal = Here, the slippage tolerances are assigned by random sampling over a truncated normal distribution with $\text{mean}(\mu) = 0.25$, $\text{standard deviation}(\sigma) = 0.25$, $\text{lower bound} = 0.1$, and $\text{upper bound} = 5$.
- Loose = Here, the slippage tolerances are assigned by random sampling over a truncated normal distribution with $\text{mean}(\mu) = 0.75$, $\text{standard deviation}(\sigma) = 0.75$, $\text{lower bound} = 0.1$, and $\text{upper bound} = 5$.

Only one of these two is used at any given time to run a simulation. This variation in the slippage tolerances allows us to study the sensitivity of different users to the changes in the simulated market conditions.

2) Urgency: Each user is also assigned an urgency factor, $\upsilon$, to modify their slippage tolerance, $\tau$.

$$\tau = \tau_{t-1} \cdot e^{\upsilon}$$

A user with a high $\upsilon$ indicates that they need to make the swap urgently, and they are less likely to wait for the market conditions to change in hopes of obtaining a better price. On the other hand, a user with a lower $\upsilon$ is more likely to wait out the market - till they get the slippage they are looking for.

These urgency factors are assigned to the user objects by random sampling over a truncated normal distribution with $\text{mean}(\mu) = \ln(1.5)$, $\text{standard deviation}(\sigma) = 0.25$, $\text{lower bound} = \ln(1)$, and $\text{upper bound} = \ln(2)$.

D. Swaps

1) Generation: Swaps are generated by the environment at the beginning of each epoch, when the setCurve() method is called. Each swap object consists of,

- tolerance($\tau$) = this denotes the slippage tolerance that the user undertaking this swap will have. Once one of the aforementioned two modes has been chosen for the simulation, random samples are drawn from the chosen distribution and allotted to each swap.
urgency(υ) = this denotes the urgency factor that dictates the amount of leeway a user will allow for in their assigned tolerance.

userNum = this value is used to assign the swap to a user in the system. It is the same as the user’s index in the 0-indexed user list.

The following member variables are only used if the user “holds” the transaction and is pushed back in the swap queue.

• amt = this denotes the amount that the user chose in the first iteration of querying the protocol for the given swap.
• idx = this denotes the token index that the user chose in the first iteration of querying the protocol for the given swap.
• tries = this is a counter variable that tracks the number of queries made to the protocol with regard to the given swap.
• new = this is a flag variable that denotes whether the swap is one that was pushed back into the queue prior to the current query. If set to False, this is a returning swap, and all the above variables need to be used for the swap.

2) Handling Swaps: Each swap is randomly assigned to a user from $U = \{U_1, U_2, ..., U_N\}$, via a discrete uniform sampling and added to the swap queue, $E$. For our simulations, each epoch consists of 400 steps. The swap to be executed in a given step is decided by the swap object, $E_i$, at the head of queue $E$. The user specified by $E_i$ attempts to make the swap - if successful, the user returns status code 1; if holding, the user returns status code 0; and if the swap fails, the user returns status code -1. $E_i$ is then popped out of queue $E$.

E. Metrics

1) Evaluation Metric: We model the agent to optimize for fees. The evaluation of the agent’s performance toward this goal can be done by looking at two key factors, (1) the total fee accrued over an epoch and (2) the number of swaps canceled in an epoch. Our reward function incorporates both these factors, and we look at the sum of all rewards awarded to an agent over an epoch. We can evaluate each agent based on the change in this value.

2) Baseline: We use the same environment to generate our baseline performance. For this purpose, there is no agent, no actions are taken, and the environment operates statically with a fee rate of 0.17% and a leverage coefficient of 42.

F. Simulation

In our simulations, we thoroughly assess the performance of each agent across various slippage tolerance modes and compare it to the corresponding baseline performances. Agent 1 demonstrates exceptional proficiency when fee rates predominantly influence slippage, which is particularly evident when the swap amounts are relatively small compared to the total pool liquidity. Conversely, Agent 2 excels in optimizing fees for larger swap amounts, where the curve’s behavior significantly impacts slippage due to the deviation from its constant sum characteristic.

Integrating the strengths of both agents, QLAMMP emerges as a true breakthrough, consistently outperforming both Agent 1 and Agent 2 in the specified tasks. Its dynamic policy adjustments, affecting both fee rate and leverage coefficient, allow QLAMMP to adapt optimally to diverse market conditions and user behaviors, leading to superior fee collection while minimizing the number of canceled transactions.

To test QLAMMP’s adaptability further, we evaluate its performance under time-varied user behavior. By modifying the users’ slippage tolerance midway through epochs, we assess QLAMMP’s agility in adjusting its policy to optimize fees for different user types.
VII. RESULTS

The performance of the proposed agent, QLAMMP, is assessed with respect to the two modes of slippage tolerance, a high liquidity swap condition and a varying user demand condition.

A. Varying Slippage Tolerance

We assess the agents’ performance against a system of users with normal and loose slippage tolerances, respectively. Row 1 of Fig. 2 shows the agents’ behavior against users with a normal slippage tolerance. Row 2 of Fig. 2 shows the agents’ behavior against users with a loose slippage tolerance. Both conditions corroborate the hypothesis that Agent 1 cannot make a significant difference in total rewards because the price impact, $\phi$, is primarily governed by the fee rate, $R_f$.

The hypothesis is further concretized by Agent 2’s performance, which is significantly better than the baseline and Agent 1. This boost in rewards can directly be attributed to the control Agent 2 has over the fee rate, $R_f$. QLAMMP performs significantly better than the baseline, and we can ascribe its success to the $R_f$-modifying subset of its action set. The slight improvement of QLAMMP over Agent 2 is likely due to its additional control over the leverage coefficient, $A$.

B. High Liquidity Swap

We also assess the agents’ performance in conditions where each user swaps a relatively large fraction of tokens compared to the liquidity in the pool. Table I shows the agents’ performance when these users comprise the environment. Compared to the earlier simulation, the users now make trades of up to $18,000 in initial pools of $20,000. As hypothesized, Agent 1 outperforms the baseline and Agent 2 because as the pool state moves to a point closer to the axes, the price impact, $\phi$, increases and is governed primarily by the leverage coefficient, $A$. Agent 2 lacks control over $A$ and thus cannot adapt sufficiently to the users’ demands. Once again, QLAMMP significantly outperforms the baseline, Agent 1, and Agent 2 due to its larger action space - controlling both $A$ and $R_f$.

| Agents | Running Average | Third Quartile |
|--------|-----------------|----------------|
| Baseline | 2521.57 | 2883.13 |
| Agent 1 | 3367.88 | 3952.25 |
| Agent 2 | 2987.40 | 3707.82 |
| QLAMMP | 4007.31 | 5173.68 |

C. User Behavior Change

QLAMMP also performs well when user behavior changes from a normal slippage tolerance to a loose slippage tolerance and vice-versa. We observe that QLAMMP can quickly adapt to the new market conditions and update the protocol parameters to extract fees from the users optimally. Fig. 3 shows QLAMMP adapting its policy to suit the new market conditions.

D. Other Findings

1) Swap Size: We simulate the agents using a range of swap amounts between $1,000 and $18,000, both inclusive. Our hypothesis is based on the assumption that the agents’ performance varies across swap sizes. This hypothesis is confirmed by Fig. 4a. We see that for smaller amounts, Agent 2 has superior performance, and beyond $3,750, Agent 1 takes over and begins outperforming Agent 2. As expected, QLAMMP consistently outperforms both Agents 1 and 2.

2) Slippage Tolerance: Fig. 4b shows the rewards varying with slippage tolerance. At low tolerances, Agent 1 performs better, while Agent 2 dominates at higher tolerances.

3) Update Interval: Fig. 4c illustrates the rewards varying with the update interval. QLAMMP is more robust to changes in the update interval compared to the baseline.

4) Standard Deviation: Fig. 4d shows the standard deviation of actions across different scenarios. QLAMMP exhibits lower variability in its actions, indicating more stable performance.

Fig. 3: Row 1 shows the user behavior changing from loose to normal slippage tolerances. Row 2 shows the user behavior changing from normal to loose slippage tolerances.

Fig. 4: Other Findings
2) **Slippage Tolerance:** We also simulate the agents using a range of slippage tolerances samplings. These samplings are made with means and standard deviations ranging from 0.25 to 0.75, both inclusive. Our hypothesis - that the users’ tolerance affects the rewards accrued by the agents - is confirmed by Fig. 4b. We can see that all three agents can adapt the protocol to earn more rewards as the users’ slippage tolerance increases.

3) **Update Intervals:** A practical consideration for deploying such an agent on-chain requires a decision on how often the parameters will be updated. We look at the total rewards collected by the agent as a function of the update interval. As expected, an almost-linear relation is observed. The developer of such a protocol must consider a trade-off between the update interval and the agent’s efficacy. Fig. 4c shows QLAMMP’s performance as a function of the update interval, \( k \).

4) **Policy Evaluation:** Since the agent makes an exploitative choice at each decision epoch, \( t \), we look at the mean standard deviation of the agent’s actions across the epochs. Fig. 4d shows that QLAMMP’s mean standard deviation stays close to 2.64. This value indicates that the agent is adapting the protocol’s behavior to suit the current market demands rather than just sticking to a given protocol configuration for which it expects the highest reward over all the epochs.

VIII. **DISCUSSION & FUTURE WORK**

We present QLAMMP, a breakthrough RL agent optimizing fees on AMM Protocols. Our empirical results reveal its significant outperformance over static counterparts in fee collection. QLAMMP demonstrates adaptability to changing user mechanics and robustness in various scenarios. To achieve improved fee collection, frequent protocol parameter updates are required, albeit with increasing gas fees as the computation frequency rises. Nevertheless, QLAMMP consistently outperforms baselines in all cases. Consideration of offloading computation using an oracle is feasible, while our open-source implementation validates the agent’s soundness and robustness under diverse conditions.

To facilitate on-chain deployment, we propose two primary approaches.

- Deploying a Smart Contract with the Q-table stored on it. However, it would need to be frequently updated to ensure it is up to date with the current market movements.
- To solve the above problem, one could deploy an online agent off-chain, utilizing a decentralized oracle for periodic protocol parameter updates [24], [25]. The oracle’s consensus mechanism might select the median value among proposed updates.

In the future, we plan on using deep reinforcement learning to train the agent. We also plan on optimizing the agents’ computations and looking at efficient methods of on-chain deployment of the proposed agent, which are the keys to such agents’ practical usability.

**REFERENCES**

[1] Y. Wang, “Automated market makers for decentralized finance (defi),” arXiv preprint arXiv:2009.01676, 2020.

[2] J. Xu, K. Paruch, S. Crousaret, and Y. Feng, “Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols,” arXiv preprint arXiv:2103.12732, 2021.

[3] V. Mohan, “Automated market makers and decentralized exchanges: a defi primer,” Financial Innovation, vol. 8, no. 1, pp. 1–48, 2022.

[4] “The block - the first and final word in digital assets.” https://www.thetblock.com/data/. Accessed: 2022-09-26.

[5] R. S. Sutton, A. G. Barto, and R. J. Williams, “Reinforcement learning is direct adaptive optimal control,” IEEE control systems magazine, vol. 12, no. 2, pp. 19–22, 1992.

[6] C. Watkins and P. Dayan, “Technical note: Q-learning, Sutton,” Reinforcement Learning, pp. 55–68, 1993.

[7] R. Hanson, “Logarithmic markets coring rules for modular combinatorial information aggregation,” The Journal of Prediction Markets, vol. 1, no. 1, pp. 3–15, 2007.

[8] E. Herzog, G. Benartzi, and G. Benartzi, “Bancor protocol,” Continuous Liquidity for Cryptographic Tokens through their Smart Contracts. Available online: https://storage.googleapis.com/website-bancor/2018/04/ba8253-bancor_protocol_whitepaper_en.pdf (accessed on 6 June 2020), 2017.

[9] V. Buterin, “On path independence,” https://vitalik.ca/general/2017/06/14/marketmakers.html, Jun 2017. Accessed: 2022-09-26.

[10] H. Adams, N. Zinsmeister, and D. Robinson, “Uniswap v2 core.” https://uniswap.org/whitepaper.pdf, Mar 2020. Accessed: 2022-09-26.

[11] M. Egorov, “Stableswap-efficient mechanism for stablecoin liquidity,” Retrieved Feb, vol. 24, p. 2021, 2019.

[12] R. Angeris and T. Chitra, “Improved price oracles: Constant function computation using an oracle is feasible, while our open-source implementation validates the agent’s soundness and robustness under diverse conditions.

[13] B. Krishnamachari, Q. Feng, and E. Grippio, “Dynamic curves for decentralized autonomous cryptocurrency exchanges,” arXiv preprint arXiv:2101.02778, 2021.

[14] S. Wang and B. Krishnamachari, “Optimal trading on a dynamic curve automated market maker,” in 2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–5, IEEE, 2022.

[15] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, “Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing,” IEEE Transactions on Vehicular Technology, vol. 68, no. 8, pp. 8050–8062, 2019.

[16] M. Liu, Y. Teng, F. R. Yu, V. C. Leung, and M. Song, “Deep reinforcement learning based performance optimization in blockchain-enabled internet of vehicle,” in 2019-2019 IEEE International Conference on Communications (ICC), pp. 1–6, IEEE, 2019.

[17] O. Sattarov, A. Muminov, C. W. Lee, H. K. Kang, R. Oh, J. Ahn, H. J. Oh, and H. S. Jeon, “Recommending cryptocurrency trading points with deep reinforcement learning approach,” Applied Sciences, vol. 10, no. 4, p. 1506, 2020.

[18] C. Hou, M. Zhou, Y. Ji, P. Daian, F. Tramer, G. Fanti, and A. Juel, “Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning,” arXiv preprint arXiv:1912.01798, 2019.

[19] A. Z. Al-Marridi, A. Mohamed, A. Erbad, and M. Guizani, “Smart and secure blockchain-based healthcare system using deep q-learning,” in 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), pp. 464–469, IEEE, 2021.

[20] M. Mani, S. Phelps, and S. Parsons, “Applications of reinforcement learning in automated market-making,” in Proceedings of the GAIM: Games, Agents and Incentives Workshops, Montreal, Canada, pp. 13–18, 2019.

[21] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, “Uniswap v3 core,” Tech. rep., Uniswap, Tech. Rep., 2021.

[22] G. Van Rossum and F. L. Drake Jr, Python tutorial. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” arXiv preprint arXiv:1606.01540, 2016.

[24] L. Ma, K. Kaneko, S. Sharma, and K. Sakurai, “Reliable decentralized oracle with mechanisms for verification and dispute,” in 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), pp. 346–352, IEEE, 2019.

[25] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juel, F. Koushanfar, A. Miller, B. Magauran, D. Moroz, et al., “Chaitinlink Labs: Next steps in the evolution of decentralized oracle networks.” Chaitinlink Labs, 2021.