
Mitigating Volatility in P2E through Allowance Verification in Decentralized Platform

Md Shariful Islam^{1,†} and Hye-Young Kim^{2,*}

¹*GCT-Lab, Games Engineering Department, Hongik University, Sejong, South Korea*

²*Hongik University, Sejong, South Korea*

E-mail: sharif@mail.hongik.ac.kr; hykim@hongik.ac.kr

**Corresponding Author*

†First Author

Received 02 April 2026; Accepted 15 July 2026

Abstract

Play-to-Earn (P2E) game economies integrated with Automated Market Maker (AMM) protocols face persistent sustainability challenges driven by whale-induced price volatility and the absence of effective token-burning mechanisms. These issues erode economic fairness and discourage long-term participation within web-based blockchain gaming platforms. This paper proposes EG-AMM (Epoch-Guarded Automated Market Maker), a software engineering framework that adds a governance-aware volatility protection layer to existing AMM-based P2E infrastructures without modifying the underlying pricing model. Inspired by circuit breaker mechanisms in traditional finance, EG-AMM enforces per-epoch dynamic swap allowances for individual players, computed from price impact tolerance and configurable through Decentralized Autonomous Organization (DAO) parameters. The framework employs Merkle-proof-based cryptographic verification and on-chain accounting to ensure allowance limits are transparent and tamper-resistant. Unlike existing pool-level approaches such as static transaction caps or dynamic fee schedules, EG-AMM operates at the individual player

Journal of Web Engineering, Vol. 25.7, 1383–1418.

doi: 10.13052/jwe1540-9589.2575

© 2026 River Publishers

level, curbing whale dominance while preserving equitable market access for ordinary players and newcomers. Preliminary analysis and game-theoretic simulation confirm improvements in price stability, participation fairness, and economic resilience, offering a practical and deployable path toward sustainable token ecosystems in blockchain-integrated gaming applications.

Keywords: P2E, AMM, tokenomics, swap allowance, Merkle root.

1 Introduction

The Play-to-Earn (P2E) model has introduced a novel intersection of blockchain, digital ownership, and gaming by enabling players to earn tokens or digital assets with real-world value. Research indicates that blockchain gaming has transformed from a recreational activity to a profit-seeking business venture, yet understanding factors influencing sustained player participation remains limited [1]. Despite rapid growth, the long-term sustainability of P2E ecosystems remains challenged by volatility from large-scale actors, often termed “whales”, whose disproportionate transactions create price shocks that destabilize in-game economies [2, 3]. This paper presents EG-AMM (Epoch-Guarded Automated Market Maker) as a practical engineering integration into existing P2E AMM implementations rather than a theoretical redesign, focusing on deployable mechanisms to curb whale dominance while preserving normal gameplay economics.

Traditional automated market makers (AMMs) operating within these ecosystems have attempted various mitigation strategies. These systems employ constant product formulas ($x \cdot y = k$) as foundational mechanisms for price discovery and liquidity provision, where reserve balances determine exchange rates [4]. However, existing AMM implementations remain primarily pool-parameter based rather than player-specific, operating at transaction levels where sophisticated actors can bypass restrictions through account multiplication or coordinated trading strategies [5]. EG-AMM builds directly on these proven AMM primitives by adding an allowance computation layer that can be integrated with minimal contract modifications.

Several P2E platforms have demonstrated a critical need for more sophisticated economic safeguards. Comprehensive analysis of blockchain games reveals that business models fundamentally differ from traditional games due to blockchain-enabled asset ownership, yet current implementations suffer from inadequate volatility controls [6, 7]. Market manipulation evidence spans major cryptocurrencies, with studies showing that whale users possess

disproportionate influence over price movements through coordinated trading activities and social media manipulation [2]. Research demonstrates that large holders systematically increase positions before price increases while smaller holders reduce holdings, creating systematic disadvantages for retail participants [8]. These patterns highlight fundamental inadequacies in traditional economic models that fail to account for power imbalances inherent in token-based gaming economies [9].

In traditional finance, markets use circuit breakers to pause or slow trading when prices move too fast and too far. These pauses help calm panic selling, give participants a chance to reassess, and keep markets stable after big trades or cascading sell-offs [10]. Our research, inspired by the circuit breaker finance model, uses a similar idea in Decentralized finance (DeFi): Instead of halting the whole market, it gives each player a limited allowance per epoch. If a player reaches their limit, their trades pause until the next epoch. This spreads out large withdrawals and prevents whales from destabilizing P2E economies.

2 Related Work

2.1 Liquidity and Price Stabilization in DeFi

AMMs, on decentralized exchanges, DeFi protocol that enable trading of tokens by utilising liquidity pools that reprice automatically according to trades [11]. Unfortunately AMMs face issues with impermanent loss and slippage, that affects trades and the revenue from provision of capital when markets are highly dynamic. Conversely, Curve has a good solution in trading stablecoins utilising a hybrid bonding curve, which minimises the slippage for large trades and reallocates fees based on changing markets [12]. Balancer introduces a strategy based on weighted pools with different trading fees which avoids shocks from loss of liquidity, and offers flexibility with financing but is only at the pool level unlike targeting specific traders [12]. A game-theoretic approach suggests that small providers utilise their entire capital budget, whilst richer participants diversify their investment into numerous pools [13]. Other approaches utilize machine learning to produce predictions about future liquidity requirement, rebalancing their reserves of individual assets accordingly, e.g. with reinforcement learning and predictive AMMs [12]. Dynamic fee management offers ways of optimising pool volatility, adjusting trading fee based on imbalance or expected future pool price volatility, [14]. Similarly Machine learning fee schedule offers mitigation of risks associated with pool manipulation by whales [15].

In summary all methods result in more liquid and more responsive and well performing markets but most methods target the response of the market, pool and individual asset. None address the individual trade behaviour, particularly large trading whales. There is no control mechanism to discriminate between fair use and manipulative trading by large holders. This is a significant flaw for the health of any gaming economy which relies heavily on the balance between players and asset values.

2.2 Whale Level and Transaction-level Controls in GameFi

Whales – large token owners – often pose one of the most serious issues in decentralized gaming economics, acting as potential disruptors of both value and fairness. Decentralized games lack appropriate ways to control whales in a way that is standard in established financial markets, posing a risk to overall value and sustainability for players involved in P2E mechanics. GameFi scholarship is actively investigating how to tackle the problem using tokenomics and incentives designed around player roles. [16] Additional work is focused on how scarcity in tokenomic designs and external incentives help to keep P2E tokens stable [17] or the best way to utilize a smart contract for whale detection and stability. [18]. Both viewpoints suggest that in-depth, integrated tokenomic structures are needed to thwart their activities.

While the research on decentralized governance goes far beyond tokens with schemes like quadratic voting designed to lessen concentration of voting power in Decentralized Autonomous Organization (DAOs) [19]. Even the use of stablecoin Decentralized exchanges (DEXes) such as Mento which place caps on trade sizes, and which can pause activity during market volatility to decentralize volume control [20]. However governance solutions like the above may be adjusted for GameFi tokens to make dominance more expensive, and temporal in nature to ensure they translate into transaction level controls in games. With a transition in research from the description of token markets to prescriptive governance-led solutions, it now becomes clear that a more tangible application for enforced mechanisms can be built into AMMs within GameFi, transferring lessons from decentralization to gaming.

2.3 Circuit-breaker Mechanisms in Financial Markets

Circuit-breakers-rules that halt trading upon large price movements-have been found to decrease both volatility and panic selling. Goldstein and Kavajecz demonstrate that although prices become more stable during halts, the “magnet effect”-in which orders are clustered just before halts-actually

increases market disruption [21]. In another study, the authors investigate the performance of European exchanges with circuit breaker mechanisms, concluding that although shortening the halt interval decreases intraday volatility, highly rigid thresholds may negatively affect liquidity, delay price discovery. Therefore, careful design is needed between halt duration and threshold. [22]. Applying this model to AMM protocols can help to increase price stability, minimize the cost of slippage, and prevent resource depletion by dominant parties against new users. Enforced allowance via cryptographic Merkle tree commitment increases the robustness and decentralization of the system.

3 System Design and Implementation

In an effort to boost the explainability of designs, we built a simple game prototype to model token collapse behaviour to serve as a base case before integrating mechanism design. We implemented an EG-AMM (on Unity3D and Solidity to facilitate quick prototyping) that includes epoch-guarded allowances, Merkle-verified limits, and dynamically adjusting boosts for layered volatility dampening. The preliminary contracts were deployed on Ganache to tune mechanism parameters before full verification on the Sepolia testnet in a simulated, real-world environment.

3.1 Overview of minimal design

Our game system is comprised of three module as shown in Figure 1: wallet interface for the game player, game client and smart contracts. First, player choosing one of their wallet address to join the game. After connecting

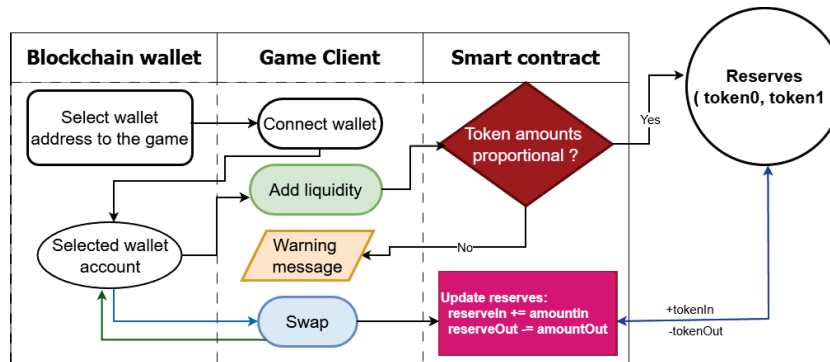


Figure 1 Architecture of minimal game design.

STORAGE

```

{ 5 items
  token0 : address "0xbC8594b84e8Dd9C2Ee . . ."
  token1 : address "0xAF4264Aa91309ce961 . . ."
  reserve0 : uint 6314492452354540000
  reserve1 : uint 1210906462065551510
  lpToken : "0xe9fB654f3Cb029fD6D7615317bBdBa02A2262b67"
}

```

Figure 2 Blockchain storage of minimal game design.

successfully the wallet address is verified for the identity and permission. For the game development, we used five player address as $A = \{a_1, a_2, \dots, a_5\}$ to perform the test cases. The smart contracts has the storage fields as illustrated in Figure 2 after connecting successfully to the wallet.

Smart contract's storage schema supports safe governance, management of tokens and liquidity in the optimized gas efficient manner by utilizing strategically packed data. It contains two immutable pointers of **token0** and **token1** with Solidity type **address** that represent the pair of ERC-20 token for swap and liquidity usage¹. The **lpToken** of type **address** stores the address of ERC-20 liquidity provider token which is responsible of creating new shares depending on liquidity provided which represents player's stake on their share. For this case we assume a scenario of a single LP in the pool. Lastly, the packed variables **reserve0** and **reserve1** of type **uint112** stores on-chain reserves of **token0** (x) and **token1** (y) and these values are updated when new liquidity is added and during the swap process to keep the invariant $x \cdot y = k$. LP will provide the liquidity by supplying proportionally according to the conventional AMM setup, therefore, innovations are primarily regarding the Algorithm. 1 by securing it with a mechanism against malicious whale actions, providing fair opportunities to everybody.

¹For experimental purpose an ERC-20 token standard framework was used to support reproducible token crashes by setting **token0** as in-game currency and **token1** as external world asset

3.1.1 Algorithm 1: Token swap implementation

The swap operation follows an interaction pathway where players initiate swap requests through the game client, which are subsequently executed by the smart contract. The swap mechanism implemented by Algorithm 1, based on the simplified constant product formula ($x*y=k$) [23]

Algorithm 1 Token swap protocol

Require: $amountIn, isToken0To1, reserve0, reserve1$

Ensure: $amountOut$ or Transaction rejection

```

1: Input validation:
2: if  $amountIn \leq 0$  then
3:   return "Insufficient input amount"
4: end if
5: let  $(reserveIn, reserveOut) \leftarrow$  if  $isToken0To1$  then  $(reserve0, reserve1)$ 
   else  $(reserve1, reserve0)$ 
6: if  $reserveIn \leq 0 \vee reserveOut \leq 0$  then
7:   Execute swap operation:
8:   if  $isToken0To1$  then
9:     Transfer  $amountIn$  of token0 from user to contract
10:     $amountInWithFee \leftarrow amountIn \times 997$ 
11:     $numerator \leftarrow amountInWithFee \times reserve1$ 
12:     $denominator \leftarrow (reserve0 \times 1000) + amountInWithFee$ 
13:     $amountOut \leftarrow numerator / denominator$ 
14:    Transfer  $amountOut$  of token1 from contract to user
15:     $reserve0 \leftarrow reserve0 + amountIn$ 
16:     $reserve1 \leftarrow reserve1 - amountOut$ 
17:   else
18:     Transfer  $amountIn$  of token1 from user to contract
19:     $amountInWithFee \leftarrow amountIn \times 997$ 
20:     $numerator \leftarrow amountInWithFee \times reserve0$ 
21:     $denominator \leftarrow (reserve1 \times 1000) + amountInWithFee$ 
22:     $amountOut \leftarrow numerator / denominator$ 
23:    Transfer  $amountOut$  of token0 from contract to user
24:     $reserve1 \leftarrow reserve1 + amountIn$ 
25:     $reserve0 \leftarrow reserve0 - amountOut$ 
26:   end if
27: end if
28: return  $amountOut$ 

```

Following the input validation, the smart contract increments its outbound token amount based on the swap logic and decrements its inbound token amount to modify its reserves. All of this happens within a secured environment with assets held by the blockchain wallet that authorizes all transactions on behalf of the players and with the game client acting as a central conductor that turns a player request into an on-chain transaction.

3.2 Advancements: EG-AMM

The EG-AMM model enhances allowance management with per-address swap volume limits in epochs (Figure 3). Innovations of the system mainly focus on tamper-resistant, player-specific swap controls.

At its core, the inherited **Ownable** mechanism designates a single administrative address (**_owner**, Solidity type **address**)—typically governed by a DAO—empowered to update governance parameters including **epochDuration** and boost configurations. The foundational AMM components (**token0**, **token1**, **lpToken**, **reserve0/reserve1**) remain invariant. The **epochDuration** parameter (Solidity type **uint256**) delineates time windows for allowance resets. Per-epoch usage is tracked in the **usedBy** mapping (Solidity type **mapping(uint256 $\mapsto$ mapping(address $\mapsto$ uint256))**), accumulating swap volumes until the allowance depletes or the epoch advances. Allowance computation integrates a volatility-derived base limit—governed by **impactToleranceBps** (Solidity type **uint16**)—with configurable boosts selected via **extraLogic** (Solidity type **enum ExtraLogicType BalanceBoost, FlatBoost**). Balance-proportional boosts apply via **balanceBoostBps** (Solidity type **uint256**), while flat allocations use **flatBoostAmount** (Solidity type **uint256**). The final allowance caps at the player's **token0** balance, ensuring feasibility. Epoch commitments employ a **merkleRootOf** mapping (Solidity type **mapping(uint256 $\mapsto$ bytes32)**), storing a single cryptographic root per epoch generated client-side from (**epochId**, **address**, **allowance**) leaves.

In technical terms, EG-AMM is basically an attempt to achieve the effect of a decentralized financial circuit breaker. Instead of pausing all trading

```
{ 13 items
  _owner : address "0x4aB5E0D87B8f27036a..."
  token0 : address "0x948d8d1083ac640090..."
  token1 : address "0x5A21aA0e64FF527FE5..."
  reserve0 : uint 2242041302112400000
  reserve1 : uint 2242041302112400000
  lpToken : "0xec29f70135BFbAC52AFA9405E99336e05B85A260"
  impactToleranceBps : uint 64
  extraLogic : {"type":"ExtraLogicType","value":"ExtraLogicType.BalanceBoost"}
  balanceBoostBps : uint 160
  flatBoostAmount : uint 57227643363100000
  merkleRootOf : {} mapping 0 items
  usedBy : {} mapping 0 items
  epochDuration : uint 132
}
```

Figure 3 Blockchain storage of EG-AMM design.

Algorithm 2 EpochId creation**Require:** epochDuration (default 5min=300s), block.timestamp (current blockchain time in seconds)**Ensure:** currentEpochId1: **Initialization:**

2: Set epochDuration = 5*60s (constructor default)

3: **Compute current epoch:**4: currentEpochId $\leftarrow \lfloor \frac{\text{block.timestamp}}{\text{epochDuration}} \rfloor$ 5: **DAO adjustment (Governance-controlled):**6: **while** DAO approves a new duration **do**7: epochDuration $\leftarrow$ newEpochDuration8: **end while**9: **Return:**10: **return** currentEpochId**3.2.2 Dynamic allowance computation framework**

Algorithm 3 computes allowances, a blend of underlying price impact based protection with discretionary boost options, all adjusted via parameters controlled by the DAO in order to calibrate economic incentives and overall stability. It first determines the base allowance from price impact limitations and then adds any player-specific boosts [25]. The constant product market maker invariant [23] can be used to set safe trade limits for swaps, and it sets the initial base allowance based on levels of volatility, moderated by DAO governance settings like impactToleranceBps (which caps the size of a single swap as a fraction of the available tokens, e.g. A 100 basis point limit on a swap to a token means up to 1% of the pool of tokens can be used, thereby not having an extreme impact on prices). Other, more incentive-based, allowances like balanceBoostBps, flatBoostAmount are available for rewarding users for participating more, thus, players with higher amounts of tokens or tokens on loan will receive higher allowances. This setup ensures balance and stability while rewarding and encouraging participation, much like a scholarship model [7] does for a P2E platform.

Volatility thresholds are set programmatically by way of DAO: e.g., they could be impactToleranceBps. These values can adjust based on market shifts to ensure they maximize both access to liquid assets whilst also preserving stability. For token swaps, the protocol computes maximal input to be bought by the system, determined as maxReserveOut percentage to reserves with price algorithm deciding on input required based on reserveOut. The protocol uses the classic $xy = k$ of Uniswap V2 to ensure reserve is not depleted using the maximal allowable output: $\Delta y_{\max}$. If the quantity is to exceed the available reserves, then it would be reduced to 0 to avoid a division-by-zero,

Algorithm 3 Compute allowance

Require: player, isToken0To1, DAO parameters (extraLogic, balanceBoostBps, flatBoostAmount, impactToleranceBps)

Ensure: allowance

```

1: if msg.sender == DAO owner then
2:   if updateExtraLogic requested then
3:     extraLogic  $\leftarrow$  new value
4:   end if
5:   if updateBalanceBoost requested then
6:     balanceBoostBps  $\leftarrow$  new value
7:   end if
8:   if updateFlatBoost requested then
9:     flatBoostAmount  $\leftarrow$  new value
10:  end if
11:  if updateImpactTolerance requested then
12:    impactToleranceBps  $\leftarrow$  new value
13:  end if
14: end if
15:  $(x, y) \leftarrow \text{getReserves}()$ 
16: if isToken0To1 then
17:   reserveIn  $\leftarrow x$  {token0}
18:   reserveOut  $\leftarrow y$  {token1}
19:   tokenIn  $\leftarrow$  token0
20: else
21:   reserveIn  $\leftarrow y$  {token1}
22:   reserveOut  $\leftarrow x$  {token0}
23:   tokenIn  $\leftarrow$  token1
24: end if
25:  $\text{maxReserveOut} \leftarrow \frac{\text{reserveOut} \times \text{impactToleranceBps}}{10000}$ 
26: if  $\text{maxReserveOut} \geq \text{reserveOut}$  then
27:   base  $\leftarrow 0$ 
28: else
29:   base  $\leftarrow \frac{\text{reserveIn} \times \text{maxReserveOut}}{\text{reserveOut} - \text{maxReserveOut}}$ 
30: end if
31: if extraLogic == BalanceBoost then
32:   extra  $\leftarrow \frac{\text{balanceOf}(\text{player}, \text{tokenIn}) \times \text{balanceBoostBps}}{10000}$ 
33: else if extraLogic == FlatBoost then
34:   extra  $\leftarrow$  flatBoostAmount
35: else
36:   extra  $\leftarrow 0$ 
37: end if
38: total  $\leftarrow$  base + extra
39: if total > balanceOf(player, tokenIn) then
40:   allowance  $\leftarrow$  balanceOf(player, tokenIn)
41: else
42:   allowance  $\leftarrow$  total
43: end if
44: return allowance

```

as $base = \frac{x \cdot \Delta y_{\max}}{y - \Delta y_{\max}} = \frac{reserveIn \cdot \maxReserveOut}{reserveOut - \maxReserveOut}$ [26] as defined by DAO's **impactToleranceBps** parameter which typically range from 1 to 5%, this prevents reserve burnout as impactToleranceBps is less than 1. It is up to the DAO to pick adequate impactToleranceBps based on reserves of the pools and volatility of connected pools, pools with high risk may have a higher ratio compared to those of stable value. Whereas, BalanceBoostBps prioritises the long term holder, FlatBoostAmount could be a flat quantity for regular gamers; it's vital to proceed with caution at first and enhance the mechanism later by measuring slippage and by stakeholders/community voting.

3.2.2.1 Numerical verification

The numerical example in Table 1 illustrates the mathematical mechanism of the allowance formula from Algorithm 3.

Consider an AMM pool with reserves $x = reserveIn = 500$ and $y = reserveOut = 1000$, and set impactToleranceBps = 1% (100 basis point) [23] To limit trade impact, cap the maximum output by $\Delta y_{\max}(cap) = y(\text{totalAmount}) \cdot \frac{\text{impactToleranceBps}}{10^4(\text{basis point})}(\text{fraction}) = 1000 \cdot \frac{100}{10^4} = 10$ (1% of 1000) i.e., this formula limits the number of basis points of the pool that can move in one trade. Using the constant-product model $xy = k$, the input required to remove Δy units of the output asset is $\Delta x = \frac{x \cdot \Delta y}{y - \Delta y}$, hence $\Delta x_{\max} = \frac{500 \cdot 10}{1000 - 10} = 5.0505$ [26].

The pre-swap mid-price is given by the reserve ratio $P_{\text{before}} = \frac{y}{x} = \frac{1000}{500} = 2.0000$ [23]. After executing the capped trade, reserves update to $x' = x + \Delta x_{\max} = 505.0505$ and $y' = y - \Delta y_{\max} = 990$, so the post-swap price becomes $P_{\text{after}} = \frac{y'}{x'} = \frac{990}{505.0505} \approx 1.9602$. Therefore, the price

Table 1 Price impact verification ($x = 500$, $y = 1000$, impactToleranceBps = 1%)

Metric	Value
Setup: reserveIn (x)	500
Setup: reserveOut (y)	1000
Setup: impactToleranceBps	100 (1%)
Step 1: maxReserveOut ($\Delta y_{\max}$)	10
Step 2: maxInput ($\Delta x_{\max}$)	5.0505
Step 3: Price before swap	2.0000
Step 4: New reserveIn	505.0505
Step 4: New reserveOut	990
Step 4: Price after swap	1.9802
Step 5: Price Impact %	1.99%

impact is $\frac{P_{\text{before}} - P_{\text{after}}}{P_{\text{before}}} \approx \frac{2.0000 - 1.9602}{2.0000} \approx 1.99\%$. The swap of 5.05 input tokens for 10 output tokens causes not more than 2% slippage. Player can not swap more than that, which is bounding the slippage into that number. The enforced limits prevent large swaps which could have substantial price impact, thereby limiting execution effects in the low liquidity scenario. Both algorithms, epoch-duration and dynamic, have complexity $O(1)$, meaning efficient execution within blockchain gas limit constraints, and allow for access controls via crypto verification.

3.2.3 Client-side Merkle root construction

This Merkle root generation occurs in client side, enabling bulk processing of allowance data in an append-only way with full cryptographic security. In the EG-AMM architecture shown in Figure 4, this operation takes place within the “Process Merkle root” step of our game-state processing: player allowance data is pooled and processed to produce a Merkle tree root. As seen in Algorithm 4, Merkle roots are produced systematically from the individual allowance data of each player by first creating leaf nodes for each player/allowance pair. To achieve this we serialize and deterministically hash each (epoch identifier, player address, allowance) triplet.

This tree is formed in a bottom-up manner, joining sibling nodes through hashing (cryptographic) into a root hash. The lexicographic ordering of sibling nodes prevents attacks stemming from order-dependence while creating deterministic structure. Merkle tree construction requires every tree to be binary so the algorithm clones nodes with odd numbers. This process being done by client avoids storing each allowances in the smart contract. Individual allowances can still be proven on chain through proofs with the Merkle root of the entire epoch’s allowance set taking up only one slot.

After finishing the computation of the Merkle root, the DAO can commit it onto the blockchain, preventing players from being responsible for the gas fees. It may appear that the system sacrifices decentralization; however, the fairness of EG-AMM is maintained through Algorithm 3, which dictates the change of the players’ swap allowance based on their current balance and DAO actions. By submitting a Merkle root, the DAO is cryptographically proving that it has followed Algorithm 3. The system may then automatically commit after fetching the allowances.

It is important to note that, problems of centralization do not revolve around who can submit root, but more around visibility and number of diverse validators. Some mature chains like Ethereum suffer from this problem with handful of mining companies writing most blocks [27], this imply

Algorithm 4 Merkle root construction**Require:** epochId, players = { (address₁, allowance₁), (address₂, allowance₂), ... }**Ensure:** Merkle root R

```

1: Step 1: create leaves
2: for all p in players do
3:   leaf  $\leftarrow H(\text{encode}(\text{epochId}, p.\text{address}, p.\text{allowance}))$ 
4:   Add leaf to  $L_0$ 
5: end for
6: Step 2: Build Merkle tree
7:  $\text{currentLevel} \leftarrow L_0$ 
8: while size( $\text{currentLevel}$ ) > 1 do
9:    $\text{nextLevel} \leftarrow$  empty list
10:  for  $i = 1$  to size( $\text{currentLevel}$ ) step 2 do
11:    if  $i + 1 \leq \text{size}(\text{currentLevel})$  then
12:       $a \leftarrow \text{currentLevel}[i]$ 
13:       $b \leftarrow \text{currentLevel}[i + 1]$ 
14:    else
15:       $a \leftarrow \text{currentLevel}[i]$ 
16:       $b \leftarrow \text{currentLevel}[i]$  {duplicate last if odd}
17:    end if
18:    Lexicographic order:
19:    if  $a < b$  then
20:       $\text{left} \leftarrow a; \text{right} \leftarrow b$ 
21:    else
22:       $\text{left} \leftarrow b; \text{right} \leftarrow a$ 
23:    end if
24:     $\text{parent} \leftarrow H(\text{left} || \text{right})$ 
25:    Add parent to  $\text{nextLevel}$ 
26:  end for
27:   $\text{currentLevel} \leftarrow \text{nextLevel}$ 
28: end while
29: Step 3: Output final root
30:  $R \leftarrow \text{currentLevel}[0]$ 
31: return R

```

the core of DAO based Merkle root doesn't fail but a number of diverse validators fail to make up for system risks. Merkle root's algorithm runs at $O(n)$, by process each node only once bottom-down, by running $O(1)$ hashes for each levels, the max number of node can reach $2n-1$ in a full binary tree and $O(n)$.

3.2.4 Client-side proof generation

To show the interaction of EG-AMM module with swap initiation, we provided system architecture on Figure 4. Upon player's initiation for swapping tokens, the client-side creates a cryptographic proof for the allowed allowance ensuring privacy for all parties. The steps of proof generation can be seen in Algorithm 5, which begins by reconstituting a player's leaf hash, finds its

position on the Merkle tree and constructs proof with siblings on path towards root enabling smart contract to prove the existence of allowance while only seeing Merkle proof not entire dataset.

Algorithm 5 Merkle proof generation

Require: epochId, playerAddress, allowance, MerkleTree(with leaves), Leaves[]

Ensure: Proof[] (Merkle path for verification)

```

1: Step 1: Encode leaf
2: leaf  $\leftarrow H(\text{encode}(\text{epochId}, \text{playerAddress}, \text{allowance}))$ 
3: Step 2: Locate leaf index
4: index  $\leftarrow$  position of leaf in leaves
5: if index < 0 then
6:   return error "Leaf not found"
7: end if
8: Step 3: Collect sibling nodes
9: proof  $\leftarrow$  empty list
10: pos  $\leftarrow$  index
11: for level=0 to MerkleTree.levels - 2 do
12:   nodes  $\leftarrow$  MerkleTree[level]
13:   sibling  $\leftarrow (pos \text{ is even}) ? pos+1 : pos-1$ 
14:   if sibling < size(nodes) then
15:     Append nodes[sibling] to proof
16:   end if
17:   pos  $\leftarrow \lfloor pos/2 \rfloor$ 
18: end for
19: Step 4: Return proof
20: return proof

```

The algorithm for generating the proof collection uses bitwise operations that help to traverse Merkle trees by figuring out whether the siblings must be collected based on whether the node at a given level is even or odd. This method covers the complete path of the proof. The client side generation has the following benefits: (1) There is no need to have any dedicated central proof server (2) Less burden on the servers. (3) Ability to generate proofs offline with complete knowledge of the tree (4) It is typically made up of $\log_2(n)$ number of elements in an array for n participants (5) It is space and computationally efficient to store proofs on the chain. The proof collecting algorithm works with $O(\log n)$ time complexity by traversing the tree from the leaf to the root, collects the necessary siblings, at each $\log_2(n)$ levels of the tree, and has to perform constant time operations at each level.

3.2.5 Swap execution with verifiable allowance enforcement

The very last on-chain interaction of EG-AMM performs the on-chain validation of the cryptographic proofs and applies epoch-bound allowance restrictions in an atomic transaction. Algorithm 6 depicts that the swap

Algorithm 6 Swap execution with verifiable allowance

Require: amountIn, isToken0To1, allowance, proof[], epochId
Ensure: Successful swap or revert

- 1: **Step 1: Verify Merkle root**
- 2: root $\leftarrow$ merkleRootOf[epochId]
- 3: **if** root=0 **then**
- 4: **revert** “root not set”
- 5: **end if**
- 6: **Step 2: Validate proof**
- 7: leaf $\leftarrow H(\text{encode } epochId, msg.sender, allowance)$
- 8: **if** MerkleProof.verify(proof, root, leaf) = false **then**
- 9: **revert** “bad proof”
- 10: **end if**
- 11: **Step 3: Enforce epoch allowance**
- 12: already $\leftarrow$ usedBy[epochId][msg.sender]
- 13: **if** already + amountIn $\geq$ allowance **then**
- 14: **revert** “allowance exceeded”
- 15: **end if**
- 16: usedBy[epochId][msg.sender] $\leftarrow$ already+amountIn
- 17: **Step 4: Optional safe cap**
- 18: **if** isToken0To1 = true **then**
- 19: maxIn $\leftarrow$ computeAllowance(msg.sender) [Algorithm 4]
- 20: **if** amountIn $\geq$ maxIn **then**
- 21: **revert** “above safe cap”
- 22: **end if**
- 23: **end if**
- 24: **Step 5: Perform constant-product swap**
- 25: approve(tokenIn, swapContract, amountIn) j- Client calls IERC20.approve
- 26: tokenIn $\leftarrow$ if isToken0To1 then token0 else token1
- 27: **if** isToken0To1 = true **then**
- 28: Transfer token0 from msg.sender to pool
- 29: amountInWithFee $\leftarrow amountIn \times 997$
- 30: numerator $\leftarrow amountInWithFee \times reserve1$
- 31: denominator $\leftarrow (reserve0 \times 1000) + amountInWithFee$
- 32: amountOut $\leftarrow numerator / denominator$
- 33: Transfer token1 from pool to msg.sender
- 34: reserve0 $\leftarrow reserve0 + amountIn$
- 35: reserve1 $\leftarrow reserve1 - amountOut$
- 36: **else**
- 37: Transfer token1 from msg.sender to pool
- 38: amountInWithFee $\leftarrow amountIn \times 997$
- 39: numerator $\leftarrow amountInWithFee \times reserve0$
- 40: denominator $\leftarrow (reserve1 \times 1000) + amountInWithFee$
- 41: amountOut $\leftarrow numerator / denominator$
- 42: Transfer token0 from pool to msg.sender
- 43: reserve1 $\leftarrow reserve1 + amountIn$
- 44: reserve0 $\leftarrow reserve0 + amountOut$
- 45: **end if**
- 46: **return** success

function gets the Merkle root corresponding to the current epoch (in case of failure to do so, the transaction is revert), hashes the player's leaf according to the triple packed (epochId, msg.sender, allowance) with the specific epoch, validates it with the obtained Merkle proof against the received root, and if it doesn't succeed, the transaction is revert for "bad proof" (see Algorithm 6, Step 2), hence preventing the allowance from being used if its corresponding leaf doesn't match.

When proof verification has completed without issue, the contract checks the players cumulative usage against the usedBy mapping, if the total of new usage plus usage to-date exceeds the available amount, it reverts with allowance exceeded. The contract optionally makes a dynamic check using ComputeAllowance (Algorithm 3), which is also protected by DAO controlled allowance settings, to avoid excess reserves being burned which results in it reverting with above safe cap, then it then approves the amount of tokens that can be sent to the contract and runs the constant-product swap logic by taking in input tokens, finding output amount using the fee adjustment in formula and then updating the reserves as displayed in Algorithm 6, Step 5.

The usedBy mapping is extended during each epoch and removes expired entities as Uniswap v3 did when it managed historic liquidity. This process requires no cleanup operations (which can be costly) while leaving an immutable historical audit trail of past allowances and also ensures that the process is cryptographically verifiable and economically viable in order to achieve decentralized scalable enforcement of swap rules. The algorithm for determining allowances has a time complexity of $O(\log n)$ to calculate allowances where n is the number of leaves in the Merkle tree with Merkle proof verification being the main contributor to the algorithm's time complexity. The dominant computational bottleneck is the Merkle proof verification step, which requires $\log_2(n)$ hash computations to validate the inclusion proof from leaf to root. Other operations have a time complexity not more than $O(1)$.

3.2.6 Data pruning and clearing storage

At the start of each new epoch, after the Merkle root for that epoch has been created, Algorithm 7 prunes expired UsedBy swap history and the corresponding merkleRootOf entries for a configurable pruneWindow, preventing state bloat and still benefiting from Ethereum's storage-clearing refunds (4,800 gas per slot under EIP-3529) [28]. The default pruneWindow can be set to 2, so the contract retains both the current epoch's mapping and the

previous epoch’s mapping, and only prunes the epoch that lies just beyond this two-epoch window.

Algorithm 7 Dynamic pruning

Require: *epochId*, *addressesToPrune*[], *newWindow*(default = 2)

Ensure: Window updated (if requested) and target epoch pruned

```

1: Step 1: Update prune window (conditional)
2: if newWindow is requested then
3:   oldWindow  $\leftarrow$  pruneWindow
4:   pruneWindow  $\leftarrow$  newWindow
5: end if
6: Step 2: Compute target epoch
7: pruneEpoch  $\leftarrow$  epochId - pruneWindow
8: Step 3: Delete usage entries
9: for i = 0 to addressesToPrune.length - 1 do
10:  delete usedBy[pruneEpoch][addressesToPrune[i]]
11: end for
12: Step 4: Delete Merkle root
13: delete merkleRootOf[pruneEpoch]
14: return success
  
```

This design gives the DAO an economic incentive as they pay gas to set Merkle roots, this small incentive can compensate them a little. Maintaining two epochs in memory also provides a proof-validity window that masks the transient race condition around Merkle root updates: if a player swaps exactly while the new root is being set and their updated allowance is not yet included in the current tree, they can still be verified against the previous epoch’s Merkle root, whereas swaps that occur after the new root is committed are verified against the current epoch’s root.

4 Experimental Analysis

To evaluate the EG-AMM’s effectiveness against whale-driven volatility, we simulated token price dynamics under stressed market conditions. The experiment was deployed on the Sepolia testnet using test accounts funded with test ETHER (ETH), providing a realistic environment for executing transactions without risking real assets. We initialized the liquidity pool with 25,000 tokens on each side. To enforce the per-player cap, the DAO-governed *impactTolerance* parameter was set to 1% (100 bps). Because the limit is calculated as a function of the pool’s reserves relative to the users’ available test balances, this 1% tolerance successfully restricted each player’s swap volume to a safe bound of 200–1200 tokens per epoch (representing roughly 1–5% of the pool). This configuration reflects documented token

concentration risks [29], sustainable GameFi participation requirements, and protocol-owned liquidity practices [30]. Crucially, while our experiment demonstrates how the smart contract technically enforces these limits on players, finding the optimal parameter is an ongoing governance task; in a live system, it is the DAO’s responsibility to continuously analyze market conditions and adjust the *impactTolerance* for the best economic results. Overall, the testnet deployment validated the allowance logic, measured the associated gas overhead, and demonstrated that enforcing per-player caps can preserve price stability while supporting adequate trading volume.

4.1 Merkle root storing cost

The gas-tracing data for 100, 500, and 1000 concurrent players shows that every Merkle root storing transaction consumes essentially the same amount of gas, with an average cost of approximately 0.0000000500 ETH per epoch across all runs. Table 2 shows the amount of gas used and gas cost for a few transactions for test cases of 100, 500, and 1000 players, which specifies that each time they consume the same gas cost regardless of the player number, which also illustrates its scalability in terms of gas usage.

From Figure 5, all three bars coincide at the same value and lie on the overall mean reference line, and their empirical variance is effectively zero. This indicates that, in the current design, the marginal on-chain cost of committing a new Merkle root does not depend on the number of players, implying that EG-AMM’s per-epoch Merkle root set overhead remains constant as the player population scales.

4.2 Price volatility

To evaluate EG-AMM’s ability to mitigate whale attacks and dampen price volatility, swap operations were executed on the Sepolia testnet via a Unity3D client. For comparison, an identical liquidity pool was instantiated on already existing standard AMM Uniswap V2 using the same test tokens and reserves (25,000 units per token) in both environments. We use Uniswap v2 as the standard constant-product AMM baseline, following prior work on optimal execution and AMM design that models Uniswap v2 as the canonical $x \cdot y = k$ market maker [23].

To isolate swap-driven price dynamics from confounding effects, we conducted experiments without concurrent liquidity provisioning. This choice is deliberate: in Mainnet blockchains, liquidity additions are partially the case of Uniswap V2, it allows any amount to be swapped, making it vulnerable to an

Table 2 SetMerkleRoot Operations: Gas Consumption Across Player Populations (5 samples per group)

Transaction Hash	Gas Used	Gas Cost (ETH)
100 Players		
0x53ec0e6b09736c9450cf082091e4dbbc6abfe10bd7d165e38554ba11fdf15e6e	48,777	0.0000000500
0x28529d1332e18d423df95f1fb23e6c3ce1beeddb6fe6ead0d889770442d673fc	48,765	0.0000000500
0x9a835dd88512c52adb41d189d4aa62aabb1d3dc19d25978519aeb8e9bdd9b4ea	48,777	0.0000000500
0xe179308828c384510974094292ea7573f719bc0f10ba8a89ca31067b40482740	48,777	0.0000000500
500 Players		
0x361b7a516bfde8cccdaceecd4ec6b1a1fe418a69eb6453c4df4c4bbb00250eaa	48,777	0.0000000500
0x492e7d1f24938903a3598fbb2b434950805199b91f1b3fd836e3a4659600889	48,777	0.0000000500
0xa7c96f9e7ebb8ee3429a747fd9b625c5fd3b1e78c08235431c84ce6f63a225ed	48,777	0.0000000500
0x2148fd9bfb7e690dff30d771e875a8476ed05dc6a09f8a57ebb5c88046e065ee	48,777	0.0000000500
0x231350777434a6a572e415d3e1946b97e5a3bb18a8391c8dad0b10b096549d8d	48,777	0.0000000500
1000 Players		
0x7370bedfd79e8c86c9d39bc949b44504fdc5c3f9ae527c796b9489890627920f	48,765	0.0000000500
0x68384d9bc5252902eccd785d136f3d3fc9e4389d8ad2812ebbe158fd3e8566d1	48,777	0.0000000500
0xa7bfa92396fbb3fa6b5787b92a2e8b952ad8426c7f559a4609668b36b0d2d561	48,765	0.0000000500
0x6db2e9221808a3324c96cc9378eec3d0b38e397856633193180852ccf6b652ca	48,777	0.0000000500
0xe72522ea10b671dbb084fc3825b3e8e61f25040b926e4297ca7dc9b2c08caef0	48,777	0.0000000500

Note: All Merkle root storing operations across 100, 500, and 1000 player populations consume consistently 48,765–48,777 gas and cost 0.0000000500 ETH per transaction. This invariance in gas cost demonstrates $O(1)$ complexity independent of ecosystem scale, validating EG-AMM's scalable Merkle root commitment mechanism.

individual swap volume (not liquidity rebalancing), causing price volatility, and whether participants receive consistent execution prices during sequential trading. *Our research question is specific: Do whales cause volatile price swings while retail participants execute at unfavorable rates?* This controlled setting allows unambiguous attribution of observed price movements to swap-volume differences alone.

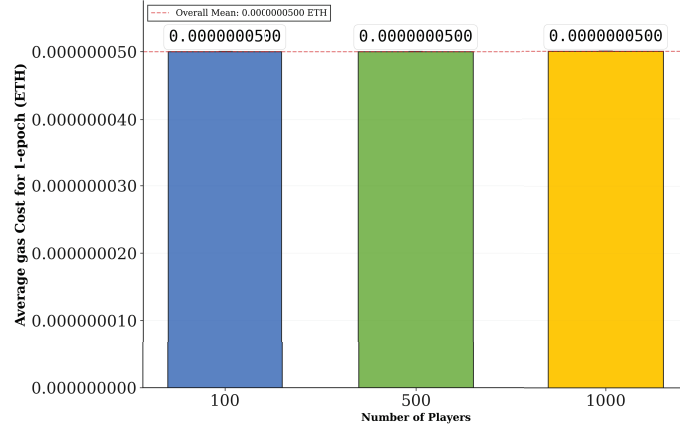


Figure 5 Merkle root storing gas cost.

Table 3 Uniswap V2 swaps demonstrating reserve depletion and price movement

TxHash	Amount	Reserve0	Reserve1	Spot price
0xff9984ede164104fd459c051699c102ef48f05126ac794e518ec3c036905ce7e	3000.00	23116.52	27095.46	0.8532
0x4d27bc5b28f8ddb1495e393deecf0005eea30a9e6453100104a23703e457633	4992.00	18956.01	33061.46	0.5734
0xd0522842dcecb94257813e83d8355af109eea5b3b6aa859bc83f684256effd9f	3000.00	13955.87	44946.46	0.3105
0x1d88e11ee48ea0e9dec99962280fcc25a90cdb51ac75d90df644304a4f037f94	5000.00	12481.45	50272.46	0.2482
0x0c784fd5ca91d96fe437a90700d4a3b94892d32a7006a8f17f8d031c48f15cd2	3000.00	11314.63	55472.46	0.2040
0xaae475e734d5f81293748321beaf296d64b1add2d601c56004d3dcd6d542fcf3	3000.00	8749.22	71792.46	0.1219

Table 4 EG-AMM swap transactions showing progressive price decline

TxHash	Amount	Allowance	Reserve0	Reserve1	Spot Price
0xdb32c2cdf8eb785401bfd265564f760384a6c871f8100aaf05b8925d9ab6432	4039	353	22665.49	27583	0.8217
0xe3baf18349f4165d7ad3487c311eb7abce7ba66f3d4bf51f3a50bd483588478	4037	383	21300.53	29356.00	0.7254
0x8122b3cc15e818190010a433830d305286c5208d08fca4ac13c47e50c39f78f0	3365	366	19777.29	31624.00	0.6254
0x2e56690beb94aac89cb8b03efa9583955c1eb80049eb32f8003f59a87a733ba	4520	436	17609.16	35530.00	0.4956
0x35a5d50a7657dc540548ca0fbc29d5f6c4477414ce14d23b95ef4b396b3f0f6	4817	483	16327.78	38327.00	0.4260
0x8b3fe39fd8409979203aea52fe60047b9246696e59434780696e2f1da890f420	3733	535	14078.84	44469.00	0.3166
0x6f36180e611bf564198fd1646fcf209051c596095005f57202618bf634d071ec	3313	576	11593.78	54032.00	0.2146
0xac8de0a61a796e48320843450b547cc5817b1fbbf4a350d26f936c33946e418d	3520	645	11184.15	56017.00	0.1997
0x50fe536d8f5e7ac7513338bd70f79967bd4c4713cefa5ffidcc7fcc81b9f3b73	3804	652	11184.15	56017.00	0.1997
0x3e73a821ca459534ded2f6fcc5a042f6bb59309f786d13d7fde444e721bae87b	4429	728	9490.51	66046.00	0.1437

4.2.1 Uniswap V2 vulnerability

Figure 6 shows the change in players' swap amounts over 90 swaps in the Uniswap V2 pool. Swaps that are far larger than normal player swaps are considered whale-attack swaps. Uniswap V2 imposes no transaction-level swap limits, permitting individual transactions of arbitrary magnitude. Table 3 documents six high-volume transactions (3000–5000 tokens each) representing whale activity. Under the constant-product formula, these large swaps induce rapid pool imbalance: the spot price ratio declines from 0.8532 to 0.1219. This uncontrolled behavior enables a single actor to cause severe price drift, constituting the classical whale attack scenario.

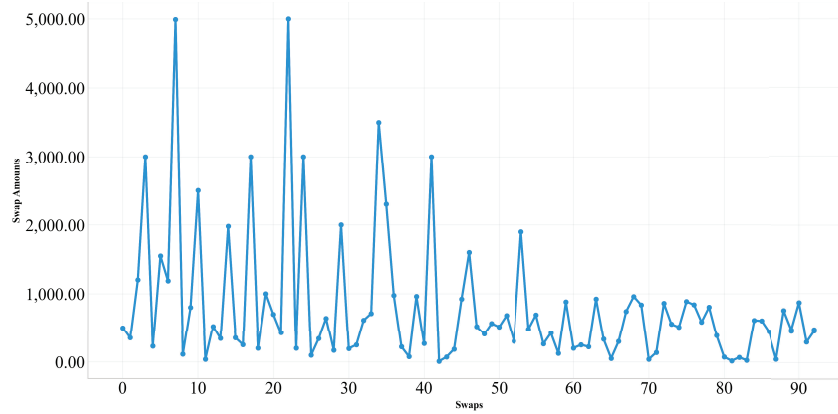


Figure 6 Uniswap v2 test swap amounts.

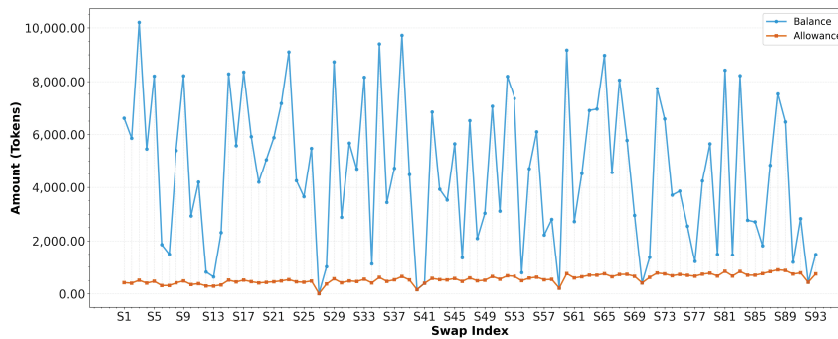


Figure 7 EG-AMM test swap amounts.

4.2.2 EG-AMM allowance mechanism

The proposed EG-AMM framework enforces per-transaction allowances independent of participant holdings. Figure 7 illustrates this constraint: player balances (3000–10,000 tokens) far exceed permitted swap amounts (300–700 tokens). This structural throttle bounds each participant's market impact regardless of financial capacity, preventing destabilizing large trades. In Table 4, we also selected several transactions matching the swap amount with the Uniswap v2 swap amount. In the case of EG-AMM, as it allows only a portion of the total amount to avoid whale attack, it is visible that reducing the exact spot price ratio takes a longer time. As we don't change the liquidity anymore, continuous swap will always change the price low, but the main inspection is how much time it is taking and during the reduction, if everyone was able to get a feasible price or not.

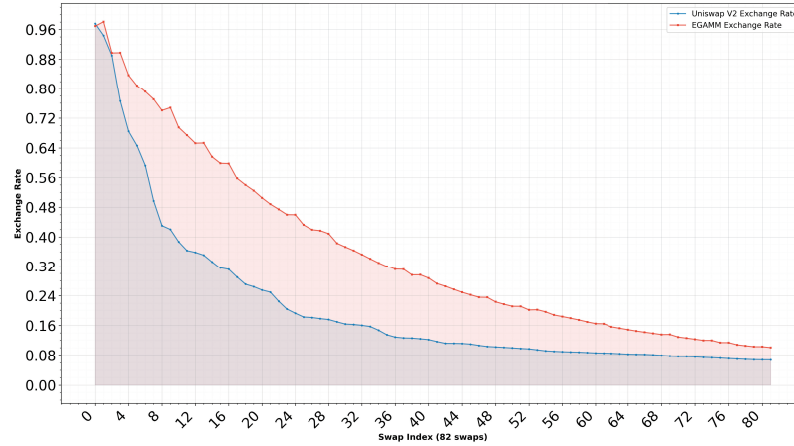


Figure 8 Price volatility: UniswapV2 vs. EG-AMM.

4.2.3 Price discovery dynamics

Figure 8 compares exchange rate trajectories across 82 sequential swaps. Uniswap V2’s spot price declines erratically from 0.97 to 0.08—indicating whale-driven volatility. EG-AMM’s price declines smoothly (0.97 to 0.10) with volatility confined to $\pm 5\%$ intervals, reflecting its allowance constraint. The key finding: in EG-AMM, participants execute consistently without severe slippage. Conversely, Uniswap V2 cost unpredictable execution prices that favor large traders.

EG-AMM’s quasi-linear price path ensures that all participants, regardless of trade timing, experience predictable slippage within a narrow band. Uniswap V2’s erratic path creates information asymmetry: trades executed during whale-driven spikes face severe losses. By capping per-transaction impact, EG-AMM distributes price risk equitably, eliminating the advantage large traders enjoy in unrestricted AMMs.

4.3 Quantitative comparison

For the quantitative evaluation of the standard AMM baseline and EG-AMM, we conducted two experimental scenarios: (1) normal players with a single whale, and (2) an equal number of normal players and whales. Using the resulting swap traces, we computed per-swap slippage [31] and the standard deviation [32] of returns for both pools (Uniswap V2 and EG-AMM deployed on Sepolia). The following subsections report and discuss the results.

Table 5 Swap outcomes with 1-whale swap for Uniswap V2 vs. EG-AMM

		Amount (tokens)			
		1000	10000	150	50
Uniswap V2	AmountOut	958.77	6663.63	71.89	23.83
	R_0	25000	26000	36000	36150
	R_1	25000	24041.24	17377.59	17305.71
	P_{ref}	1	1.08	2.07	2.09
	P_{exec}	1.04	1.50	2.09	2.10
	Slippage	4.30	38.76	0.72	0.44
EG-AMM	Allowance	277.53	505.32	150	50
	AmountOut	273.15	482.91	139.82	46.25
	R_0	25000	25277	25782	25932
	R_1	25000	24726.85	24243.94	24104.12
	P_{ref}	1	1.02	1.06	1.08
	P_{exec}	1.01	1.05	1.07	1.08
	Slippage	1.41	2.30	0.88	0.49

Note: Standard deviation of returns for Uniswap V2 is 0.504 and for EG-AMM it is 0.014.

4.3.1 3 normal players with 1 whale

Table 5 reports with 4 swaps with comparable trade sizes executed on both Uniswap V2 and EG-AMM. For each swap we record the output amount, pre-trade reserves (R_0, R_1), the reference price P_{ref} , the execution price P_{exec} , and the resulting slippage. In the Uniswap V2 baseline, a whale attempting to swap 10,000 tokens in a single transaction induces severe price impact, producing 38.76% slippage. In contrast, EG-AMM bounds the maximum per-epoch swap size via its allowance mechanism; in this trace, the effective swap input is limited to 505.32 tokens, which reduces the maximum observed slippage to 2.30%.

We further compute realized volatility from the same data return series. Under the same sequence of operations, the standard deviation of returns is 0.50 for Uniswap V2 and 0.01 for EG-AMM, corresponding to an approximate reduction of 97.1% (i.e., $\frac{0.50-0.01}{0.50} \times 100$).

4.3.2 3 whales vs. 3 normal players

Table 6 reports six swaps where three whales and three normal players execute similar trade sizes on both Uniswap V2 and EG-AMM where both pool have mostly closer size of reserves. As in the previous experiment, we log the same set of variables for each swap. In the Uniswap V2 baseline, whale swaps of 10,000, 8000, and 5000 tokens generate slippage of 40.26%, 22.82%, and 11.53%, respectively, making prices highly unstable for normal

Table 6 Swap outcomes with 3-whale vs. 3-normal player swap

Uniswap V2							
Amount	Allowance	AmountOut	R_0	R_1	P_{ref}	P_{exec}	Slippage
10000	...	7129.59	25025.39	25025.37	1.00	1.40	40.26
500	...	251.13	35025.39	17895.78	1.96	1.99	1.73
8000	...	3235.15	35525.39	17644.65	2.01	2.47	22.82
1000	...	322.67	43525.39	14409.50	3.02	3.10	2.59
5000	...	1418.34	44525.39	14086.82	3.16	3.52	11.53
150	...	38.14	49525.39	12668.48	3.91	3.93	0.60
EG-AMM							
10000	502.52	490.67	25000	25000	1.00	1.02	2.31
500	265.02	251.32	25502	24509.33	1.04	1.05	1.34
8000	452.52	416.96	25767	24258	1.06	1.08	2.06
1000	277.52	248.50	26219	23841.05	1.09	1.11	1.36
5000	377.52	329.99	26496	23592.55	1.12	1.14	1.72
150	150	128.74	26873	23262.55	1.16	1.16	0.86

Note: Standard deviation of returns for Uniswap V2 is 0.387 and for EG-AMM it is 0.009.

players holding 500, 1000, and 150 tokens at the same reference price, as execution prices increase sharply.

In contrast, under EG-AMM the whales receive bounded allowances of 502.52, 452.52, and 377.52 tokens per epoch, which limit their effective trade sizes; slippage falls to 2.31%, 2.06%, and 1.72%, values close to those observed for normal-player swaps.

We also evaluate volatility using the standard deviation of returns. After the same sequence of swaps, the return standard deviation for Uniswap V2 is 0.387, whereas for EG-AMM it is 0.009, corresponding to an approximate reduction of $\frac{0.387-0.009}{0.387} \times 100 \approx 97.7\%$.

Both quantitative analyses indicate that epoch-guarded swap allowances substantially reduce whale-driven price dislocations and stabilize short-horizon price dynamics. By bounding large trades that would otherwise destabilize the market, EG-AMM protects all participants—not only smaller players—while keeping execution prices much more consistent over time.

4.4 Swap operation gas cost difference

To assess the gas cost on swap operations relative to other already stable implementations, we conducted empirical measurements of transaction costs on both Uniswap V2 and EG-AMM.

Table 7 Uniswap V2 swap transaction gas cost

TxHash	Amount	Gas Used	GasCost(ETH)
0xfa207a296b8a2330120b0325ac79d0dec853db9a21b56b64dc0985ce4eabeda4	500	101773	0.00000011
0xff9984ede164104fd459c051699c102ef48f05126ac794e518ec3c036905ce7e	3000	101773	0.00000011
0xcbe23f8393bf46029b34c8363a81ade4ff68176ec2b4fbdebe5727f8239e8278	1000	101773	0.00000012
0x1d88e11ee48ea0e9dec99962280fcc25a90cdb51ac75d90df644304a4f037f94	5000	101785	0.00000012
0x8e6e91f818789448a9b6c9d7d424291deb8d75e200efc877df8245b966be95dc	10	101749	0.00000011

Table 8 EG-AMM swap transaction gas cost

TxHash	Amount	Gas Used	GasCost(ETH)
0xc22eca36611f45485b140acfedc3695a1808477db349bf800ed065e4e2c2fe1d	200	152948	0.00000015
0xa324003a5d675b964d40cbda874080c74a61e9468b4c8f0418edd0a1b767d769	560	152946	0.00000015
0x2a505aa789d2193556d00ad88c30c525852541ec96b9e05d6b11f1c15d99a1d2	752	152934	0.00000015
0xda434913c08797e089e799c1edb71c432f69a2af010a5323a31b812a60e7a121	12	152169	0.00000015
0x1b52433c6b2e332a9d94bcd1c03ccfa793b5842fe2d2192bf9b89feb07f54f8	160	152152	0.00000015

4.4.1 Uniswap V2 baseline

Our analysis of Uniswap V2 transactions on Sepolia testnet reveals that standard swap operations consume approximately 101 thousand gas per transaction (Table 7). This measurement aligns with documented specifications indicating that Uniswap V2 swaps incur an average cost of 102 thousand gas [33]. The consistency across both empirical and documented values validates our testing methodology.

4.4.2 EG-AMM overhead

In contrast, the same swap operation on EG-AMM costs approximately 152,000 gas on average (Table 8), representing a differential of 51,000 gas increase. As detailed in Figure 9, this overhead is entirely attributable to the mapping-based allowance mechanism. Specifically, each swap transaction includes an SSTORE (storage write) operation that records the players cumulative swap amount within the current epoch to the usedBy mapping [34]. This storage cost is necessary for the anti-whale fairness guarantee.

4.5 Public chain constraints

All public blockchains face fundamental constraints imposed by fixed block production intervals and per-block gas capacity. These parameters vary significantly across major chains: Ethereum operates at 12-second block intervals with a 30,000,000-gas limit per block [35, 36]; BNB Smart Chain produces blocks approximately every 3 seconds [37]; Polygon chain achieves 2-second block times [38]; and Solana targets 400–600 millisecond slot intervals [39]. Since our implementation was deployed on Sepolia (Ethereum testnet), all subsequent analysis assumes Ethereum’s protocol parameters.

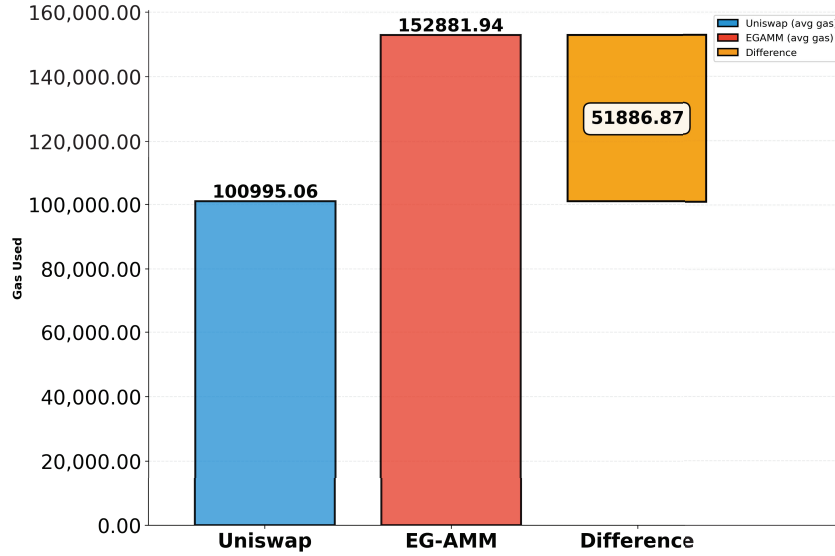


Figure 9 Gas cost difference: Uniswap vs. EG-AMM.

Table 9 Swap per-epoch capacity comparison: Uniswap V2 vs. EG-AMM

Epoch	Blocks	Gas Capacity	Max Players	
5 min	25 blocks	750M gas	Uniswap V2: 7366	EG-AMM: 4901
2 min	10 blocks	300M gas	Uniswap V2: 2947	EG-AMM: 1961

In Table 9, a standard 5-minute epoch encompasses approximately $300/12 = 25$ blocks only with the swap operation, excluding all other (Merkle root store, liquidity add) operations. The aggregate gas capacity is therefore $25 \times 30M = 750M$ gas. Under these constraints, the Maximum concurrent swap throughput for Uniswap V2 is $750M/101,000 = 7366$ swaps. On the other hand, the Maximum concurrent throughput in EG-AMM can be $750M/152,000 = 4901$ swaps. Estimatively, for a 2-minute epoch, there can be $120/12 = 10$ blocks. For 10 blocks, the aggregate gas capacity: $10 \times 30M = 300M$ gas. In that case, the Maximum concurrent swap throughput in Uniswap V2 is $300M/101,000 = 2947$ swaps and the Maximum concurrent swap throughput in EG-AMM is $300M/152,000 = 1962$ swaps.

These calculations demonstrate a fundamental trade-off inherent to EG-AMM's design. The EG-AMM imposes a 51,000-gas extra charge per transaction. As a result, practical deployment scenarios face player capacity constraints. A modest user base of 5000 concurrent players becomes harder

to exceed in the 5-minute epoch capacity under optimal conditions (assuming pure swaps and zero overhead from auxiliary operations). In contrast, targets for ten to a thousand players become infeasible. Critically, this throughput limitation is not unique to EG-AMM; it reflects a universal constraint across all public blockchains, imposed by immutable block production rates and gas limits. This represents a deliberate design choice of our EG-AMM: accepting reduced transaction throughput in exchange for provable protection against whale-driven market manipulation.

5 Limitations

The cost of updating the epoch Merkle root increases with the number of active players. This growth occurs because computing each Merkle root requires remote procedure calls (RPCs) to fetch player balances—specifically, one RPC call per player to query the account balance and determine the allowance value for each leaf in the tree. Empirically, the Merkle root storing transaction requires approximately 105.2 seconds for 100 players, 463.1 seconds for 500 players, and 911.4 seconds for 1000 players (Table 10).

This data reveals a per-player latency of roughly 0.9–1.2 seconds, indicating a nearly linear scaling relationship (Figure 10). Extrapolating from the mean per-player latency values in Table 10, if we estimate the execution time for 10,000 players at approximately 1 second per player, the total execution time would be roughly 2.7 hours (10,000 seconds). Additionally, public blockchain RPC endpoints impose rate limits that trigger errors even during concurrent read operations such as balance queries. Although public chains provide daily and per-second credit quotas, free-tier limits are quickly exceeded during intensive RPC calls, which was our primary testing concern [40]. Even public-chain documentations confirms that all chains enforce unavoidable rate limits, causing read operation execution time increase linearly with player count. This represents a scalability concern. Consequently, epochs with several hundred or more active players experience

Table 10 Merkle root execution mean time

Players	Merkleroot Execution Time(s)	Mean Per-player Latency(s)
100	105.248	1.05248
500	463.105	0.92621
1000	911.397	0.911397

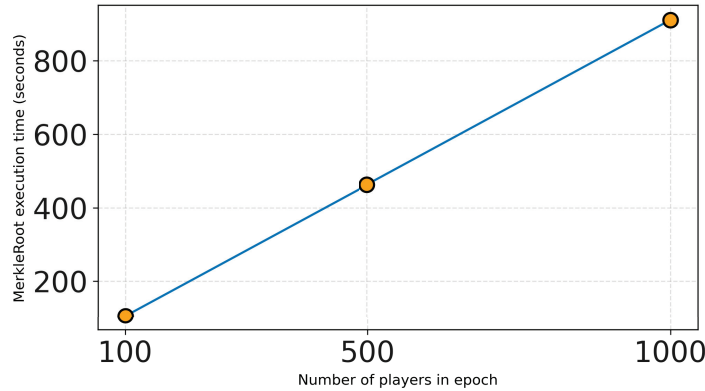


Figure 10 Merkle root creation and execution time in different number of players.

substantial finalization delays for the Merkle root computation. In large-scale deployments, such latency can be a bottleneck for system responsiveness and real-time gameplay mechanics, and thus represents a key scalability limitation of the current EG-AMM.

Additionally, EG-AMM inherits a limitation related to multi-wallet circumvention. Because swap allowances are enforced per address, a determined whale can partially bypass these limits by splitting holdings across many wallets and trading from each one. This pattern corresponds to a standard sybil strategy in crypto, where a single actor controls multiple addresses that appear as independent users or voters [41]. Although the mechanism still raises the cost of such behavior—requiring more gas and enforcing the epoch-based allowance on every address—it cannot fully prevent a motivated adversary from aggregating effective capacity through multi-wallet setups.

On the other hand, there is also a governance and parameter-setting risk. EG-AMM relies on a DAO to configure key parameters such as epoch duration, impact tolerance, and balance-based boost percentages. In token-weighted voting, decision power is often concentrated among a small number of large token holders (“whales”), which can skew outcomes toward their preferences and undermine the fairness of the mechanism. For EG-AMM, a captured or inattentive governance process could weaken whale mitigation (by raising allowances or shortening epochs) or over-constrain normal players (by setting overly strict limits), thereby reintroducing volatility or harming liquidity. Therefore, careful market analysis and robust, transparent DAO governance are prerequisites not only for EG-AMM but also for any blockchain markets.

6 Future Work

A key limitation of the current design lies in the operational overhead of constructing Merkle trees. For instance, the client must issue RPC calls per epoch according to the players' amount to gather balances. While these queries are read-only and incur no direct gas cost, they are constrained by API rate limits from public node providers and network latency, that delays merkle root generation and submission.

Future work could mitigate this challenge through testing several strategies. For example, partitioning players geographically across regional servers to compute Merkle roots in parallel may distribute the workload and substantially reduce execution time. Additionally, exploring incremental or append-only Merkle tree structures also may provide an effective solution by enabling efficient updates without full recomputation. Real-time allowance adjustments within the Merkle tree represent an additional enhancement direction. The system should further investigate how to protect against multi-wallet strategies (Sybil attacks). In this context, it is conceivable that a large whale splits their total allowance into numerous smaller wallets in order to work around per-wallet restrictions. These are areas which include establishing links between a person and a wallet (identity linking approaches, for instance, in the scope of proof-of-personhood), creating higher stake ages which would grant preferences to older wallets and implementing an on-chain cluster heuristic for discovering concerted wallets.

7 Conclusion

This study presented a framework for enforcing per-player limits in play-to-earn (P2E) economies through a combination of mapping-based tracking and Merkle tree commitments, effectively embedding a decentralized circuit-breaker within the AMM swap logic. By treating each epoch's allowance as a localized halt threshold—where any swap request exceeding the remaining per-epoch cap is suspended until the next epoch reset-EG-AMM guarantees that no participant can trigger an unbounded liquidity shock. The mapping for amount tracking enforces real-time gate checks akin to traditional trading halts, while Merkle proofs ensure scalable, permissionless validation of each players' allowance without on-chain per-user storage. Experimental validation in a testnet demonstrated feasibility for systems with hundreds of users per epoch. The main contribution lies in showing that this mapping-plus-Merkle implementation provides a practical balance between cryptographic

soundness and on-chain enforceability. By coupling time-distributed liquidity release with per-player allowance gates—mathematically mirroring circuit-breaker thresholds—EG-AMM avoids restrictive or punitive models and enhances the sustainability of token-based ecosystems.

The proposed framework can be applied by blockchain game developers, decentralized exchange designers, and DAO architects seeking scalable mechanisms for equitable resource distribution. Its principles may further extend to financial platforms and NFT marketplaces where controlled and auditable participation is required.

Acknowledgment

This research was supported by Culture, Sports and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism in 2024(Project Name: Global Talent Training Program for Copyright Management Technology in Game Contents, Project Number: RS-2024-00396709, Contribution Rate: 50%) and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2026-25492536).

Code Availability

The source code for this research is available at: [EG-AMM github repository]. As proof of work, you can see recordings of the project in these links: [minimal design-based simulation], [EG-AMM design-based simulation]

References

- [1] Y.-Y. Ku, H.-P. Lu, and K.-L. Hsiao, “Assessing the perceived risks and benefits of play-to-earn models in blockchain gaming: Integration of the PLS-SEM and ANN method,” SSRN, Working Paper 4835736, 2024, doi: 10.2139/ssrn.4835736.
- [2] W. Li, L. Bao, J. Chen, J. Grundy, X. Xia, and X. Yang, “Market manipulation of cryptocurrencies: evidence from social media and transaction data,” *ACM Transactions on Internet Technology*, vol. 24, no. 2, pp. 1–26, 2024, doi: 10.1145/3643812.
- [3] A. Chernoff and J. Jagtiani, “Beneath the crypto currents: The hidden effect of crypto “whales”,” Federal Reserve Bank of Philadelphia, Working Paper, 2024, doi: 10.21314/JCR.2025.007.

- [4] N. Dimitri, "Arbitrage in automated market makers," *Frontiers in Blockchain*, vol. 7, p. 1455070, 2024, doi: 10.3389/fbloc.2024.1455070.
- [5] Á. Cartea, F. Drissi, and M. Monga, "Decentralised finance and automated market making: Execution and speculation," *Journal of Economic Dynamics and Control*, p. 105134, 2025, doi: 10.1016/j.jedc.2025.105134.
- [6] H. Ko, S. Lee, and J. Seo, "Ethereum smart contract account classification and transaction prediction using the graph attention network," *Journal of Web Engineering*, vol. 23, no. 5, pp. 657–680, Aug. 2024, doi: 10.13052/jwe1540-9589.2353.
- [7] Y. Lai, S. Fan, and W. Cai, "Quantitative analysis of play-to-earn blockchain games: A case study of axie infinity," in *2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom)*. IEEE, 2023, pp. 250–257, doi: 10.1109/MetaCom57706.2023.00054.
- [8] W. Chen, J. Wu, Z. Zheng, C. Chen, and Y. Zhou, "Market manipulation of bitcoin: Evidence from mining the mt. gox transaction network," in *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2019, pp. 964–972, doi: 10.1109/INFOCOM.2019.8737364.
- [9] G. K. S. Tan, "Assetizing the video game: Play-to-earn (P2E) games and blockchain rentiership," *Progress in Economic Geography*, vol. 3, no. 1, p. 100036, 2025, doi: 10.1016/j.peg.2025.100036.
- [10] B. Clapham, P. Gomber, M. Haferkorn, and S. Panz, "Managing excess volatility: Design and effectiveness of circuit breakers," SAFE, Working Paper, 2017, doi: 10.2139/ssrn.2910977.
- [11] D. Kirste, A. Poddey, N. Kannengießer, and A. Sunyaev, "On the influence of conventional and automated market makers on market quality in cryptoeconomic systems," *Electronic Markets*, vol. 34, no. 1, p. 42, 2024, doi: 10.1007/s12525-024-00723-1.
- [12] T. Lim, "Predictive crypto-asset automated market maker architecture for decentralized finance using deep reinforcement learning," *Financial Innovation*, vol. 10, no. 1, p. 144, 2024, doi: 10.1186/s40854-024-00660-0.
- [13] W. Tang, R. El-Azouzi, C. H. Lee, E. Chan, and G. Fanti, "Game theoretic liquidity provisioning in concentrated liquidity market makers," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 9, no. 1, pp. 1–45, 2025, doi: 10.1145/3711700.
- [14] I. Lebedeva, D. Umnov, Y. Yanovich, I. Melnikov, and G. Ovchinnikov, "Dynamic fee for reducing impermanent loss in decentralized

- exchanges,” in *2025 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2025, pp. 1–5, doi: 10.1109/ICBC64466.2025.11114646.
- [15] N. Burton, “Mitigating risk and loss in AMM liquidity pools: A dynamic fee system based on risk prediction,” SSRN, Working Paper 5035111, 2024, doi: 10.2139/ssrn.5035111.
 - [16] A. Xie, X. Hu, M. Wang, and X. Zhao, “Exploring the sustainable development of web3 game token economy,” *Sustainability*, vol. 16, no. 15, 2024, doi: 10.3390/su16156587.
 - [17] S. Tamai and S. Kasahara, “DAO voting mechanism resistant to whale and collusion problems,” *Frontiers in Blockchain*, vol. 7, p. 1405516, 2024, doi: 10.3389/fbloc.2024.1405516.
 - [18] A. RajabiNekoo, L. Rasoul, A. Farhadi, and A. Zamanifar, “SILS: Strategic influence on liquidity stability and whale detection in concentrated-liquidity DEXs,” *arXiv preprint*, 2025, doi: 10.48550/arXiv.2507.19411.
 - [19] A. R. Duguleană, C. R. Tănăsescu, and M. Duguleană, “Emerging trends in play-to-earn (P2E) games,” *Journal of Theoretical and Applied Electronic Commerce Research*, vol. 19, no. 1, pp. 486–506, 2024, doi: 10.3390/jtaer19010026.
 - [20] Mento Protocol, “Trading limits and circuit breakers,” Mento Blog, <https://www.mento.org/blog/defi-security-mento-circuit-breaker-in-action>, 2025, [Accessed 21-09-2025].
 - [21] M. A. Goldstein and K. A. Kavajecz, “Trading strategies during circuit breakers and extreme market movements,” *Journal of Financial Markets*, vol. 7, no. 3, pp. 301–333, 2004, doi: 10.1016/j.finmar.2003.11.003.
 - [22] C. Guillaumie, G. Loiacono, C. Winkler, and S. Kern, “Market impacts of circuit breakers: Evidence from EU trading venues,” European Securities and Markets Authority, Paris, France, Report, 2020.
 - [23] H. Adams, N. Zinsmeister, and D. Robinson, “Uniswap V2 core,” White paper. <https://app.uniswap.org/whitepaper.pdf>, Mar. 2020, [Accessed 02-10-2025].
 - [24] Z. Peng, Y. Zhang, Q. Xu, H. Liu, Y. Gao, X. Li, and G. Yu, “Neuchain: a fast permissioned blockchain system with deterministic ordering,” *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2585–2598, 2022, doi: 10.14778/3551793.3551816.
 - [25] R. Han, Z. Yan, X. Liang, and L. T. Yang, “How can incentive mechanisms and blockchain benefit with each other? a survey,” *ACM*

- Computing Surveys*, vol. 55, no. 7, pp. 1–38, 2022, doi: 10.1145/3539604.
- [26] G. Angeris, H.-T. Kao, R. Chiang, C. Noyes, and T. Chitra, “An analysis of uniswap markets,” *Cryptoeconomic Systems*, 11 2020, doi: 10.21428/58320208.c9738e64.
 - [27] CoinEdition, “Ethereum’s block production centralized: 3 builders control 90%,” News article. <https://coinedition.com/ethereums-block-production-centralized-3-builders-control-90/>, [Accessed 10-12-2025].
 - [28] V. Buterin and M. Swende, “EIP-3529: Reduction in refunds,” Ethereum Improvement Proposal. <https://eips.ethereum.org/EIPS/eip-3529>, 2021, [Accessed 25-10-2025].
 - [29] Footprint Analytics, “Web3 gaming analytics for investors,” CoinGecko article. <https://www.coingecko.com/learn/web3-gaming-analytics-for-investors>, Aug. 2023, [Accessed 17-11-2025].
 - [30] N. Eliason, “Building sustainable Web3 games with owned liquidity and tokenized assets,” Blog. <https://www.nateliason.com/blog/building-sustainable-web3-games>, Jan. 2022, [Accessed 05-11-2025].
 - [31] D. Mejía, “Price slippage explained: Why it matters in trading,” Equiti article. <https://www.equiti.com/sc-en/news/trading-ideas/what-is-price-slippage-in-trading/>, 2025, [Accessed 19-11-2025].
 - [32] Teju Finance, “Reading: Return and standard deviation,” Online course. <https://vlp.teju-finance.com/courses/mod/book/view.php?id=426&chapterid=970>, [Accessed 03-12-2025].
 - [33] CPlusPlusDeveloper, “Uniswap V3 guzzles 100% more gas than V2,” Reddit, r/ethereum discussion. https://www.reddit.com/r/ethereum/comments/n5xqe8/uniswap_v3_guzzles_100_more_gas_than_v2/, 2020, [Accessed 12-12-2025].
 - [34] W. Tang, “EIP-2200: Structured definitions for net gas metering,” Ethereum Improvement Proposal, <https://eips.ethereum.org/EIPS/eip-2200>, 2019, [Accessed 23-12-2025].
 - [35] Ethereum.org, “Ethereum glossary,” Documentation, <https://ethereum.org/glossary/>, [Accessed 19-12-2025].
 - [36] T. Beiko, “London mainnet announcement,” Ethereum Foundation Blog. <https://blog.ethereum.org/2021/07/15/london-mainnet-announcement>, Jul. 2021, [Accessed 26-10-2025].
 - [37] BNB Chain, “BNB Smart Chain overview,” BNB Chain documentation. <https://docs.bnbchain.org/bnb-smart-chain/overview/>, [Accessed 03-01-2026].

- [38] Binance, “Polygon network: Scaling ethereum to build the future of Web3,” Binance Square blog, <https://www.binance.com/en/square/post/30395283362073>, [Accessed 03-01-2026].
- [39] vusirikala, “Is solana slot time 400 ms or 800 ms?” Reddit, r/solana discussion. <https://www.reddit.com/r/solana/comments/txtddl/comment/i3q53o1/>, 2021, [Accessed 03-01-2026].
- [40] MetaMask, “Infura rate limit,” MetaMask Support documentation. <https://support.metamask.io/it/develop/account/limits/>, 2025, [Accessed 07-11-2025].
- [41] Crypto Times, “Trends and challenges related to sybil resistance in public chains,” Xangle research report. <https://xangle.io/en/research/detail/1781>, Jan. 2024, [Accessed 28-12-2025].

Biographies



Md Shariful Islam received the B.Sc. degree in computer science and engineering from the University of Chittagong, Bangladesh, in 2020. He is currently a research assistant in the Game Engineering department at Hongik University, South Korea. His professional experience spans software engineering roles focusing on blockchain technology, play-to-earn game development, and scalable software system architecture. He has contributed novel research ideas and implementations to these fields, as demonstrated by his peer-reviewed publications and project collaborations.



Hye-Young Kim received the Ph.D. degree in Computer Science and Engineering from Korea University, South Korea in February 2005. During her Ph.D. studies, she focused on location management schemes and traffic modeling for mobile IPv6, cellular networks, and network mobility. She developed a network protocol for 9 years while working as a senior researcher at Hyundai Electronics. She has been working as a Full-time Professor at Hongik University, South Korea, since March 2007. Her research interests include traffic modeling, load balancing schemes, and copyright technology for digital content on blockchain and web3.