

SAMM: Sharded Automated Market Maker

Hongyin Chen ✉ 

Technion, Haifa, Israel

Amit Vaisman ✉ 

Technion, Haifa, Israel

Ittay Eyal ✉ 

Technion, Haifa, Israel

Abstract

The central challenge in addressing the security-performance trade-off in blockchain systems has moved from consensus protocols to enabling parallel execution. Various *smart contracts* (stateful blockchain programs) already benefit from parallelization. But we show that non-parallelizable operations within smart contracts remain a bottleneck. A natural approach is to apply *sharding*, that is, partitioning a *single logical service* into multiple identical parallelizable shards. However, we observe that this introduces game-theoretic issues in various applications: Participants may be incentivized to overload the system.

We focus on *Automated Market Makers* (AMMs), which enable token exchange: Traders swap tokens with the AMM for a fee, while *liquidity providers* pre-fund the AMM and receive these fees. We show experimentally that their demand will soon reach the single-thread performance limit.

We present *SAMM*, a sharded AMM system where multiple independent AMM shards operate on the same blockchain. The key challenge is ensuring SAMM's security against overloading and destabilization attacks. SAMM addresses this with careful mechanism design; we provide both game-theoretic analysis and simulation-based validation. We implement SAMM on Sui and Solana testnets, achieving 5x and 16x throughput improvements, respectively, with potential for more as blockchain parallelism advances. SAMM is deployable and has already attracted industry interest.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Security, blockchain, automated market makers, game theory

Digital Object Identifier 10.4230/LIPIcs.AFT.2026.11

Related Version *Full Version*: <https://arxiv.org/abs/2406.05568> [22]

Supplementary Material *Software (Source Code)*: <https://github.com/MountainGold/SAMM-Sui-Evaluation>, archived at `swb:1:dir:c5fa4a9c00c23d38762de20b654abbf301581ff3`

Software (Source Code): <https://github.com/MountainGold/spl-samm>
archived at `swb:1:dir:3b279df02c2cd754cbbdfa156c783c932aa68a1f`

Funding This work was supported in part at the Technion by Aly Kaufman Fellowship, IC3, the Sui Foundation, and the Avalanche Foundation.

1 Introduction

The trade-off between security and performance is a key challenge in blockchain systems. Recent advances in high-throughput consensus (e.g. [73, 28, 10]) have largely shifted the throughput bottleneck from consensus to smart-contract execution [86]. Modern blockchains like Sui [14], Aptos [49], and Solana [88] support parallel execution to address this challenge (though not Bitcoin [63] or Ethereum [84]). They automatically distribute independent transactions across multiple execution threads.

Practical smart-contract deployments adopt *sharding* [89, 46, 82], a classical throughput-scaling technique, to improve performance (§2). They trivially separate independent components of a contract into multiple independent instances called *shards*, and the underlying



© Hongyin Chen, Amit Vaisman, and Ittay Eyal;
licensed under Creative Commons License CC-BY 4.0
8th Conference on Advances in Financial Technologies (AFT 2026).

Editors: Aggelos Kiayias and Maria Kyrpoulou; Article No. 11; pp. 11:1–11:27

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

blockchain runs them in parallel threads. However, non-parallelizable operations within high-demand smart contracts remain a bottleneck. To the best of our knowledge, no prior work studies improving the throughput of non-parallelizable smart contracts via sharding.

In this work, we study the sharding of a *single non-parallelizable logical service* into multiple *identical* shards. For sharding to be effective, participants' demand should be evenly distributed across shards, without incurring additional overhead from splitting a single activity across multiple shards. But in decentralized systems we cannot assume participants behave benignly; security issues arise if participants deviate from the desired behavior.

We find that in several applications, such as *lending protocols* [61, 45, 35] and *centralized limit order books* (CLOBs) [74, 31], rational participants may be incentivized to launch overloading attacks: splitting their activity across multiple shards can increase their revenue (see the full version of our report [22]). This is not merely theoretical: In practice, users already split orders across multiple similar (but non-sharded) contracts to optimize revenue via so-called *routers* [8, 93].

The root cause of this security challenge is common in blockchain protocols [90, 43, 60, 19, 87] and applications [78, 56, 27, 21, 23, 62]: They are open to anyone and participants attack the system for more revenue. The resilience of the sharded contract stems from game-theoretic security [16]: Mechanism design prevents such attacks by eliminating additional revenue due to misbehavior.

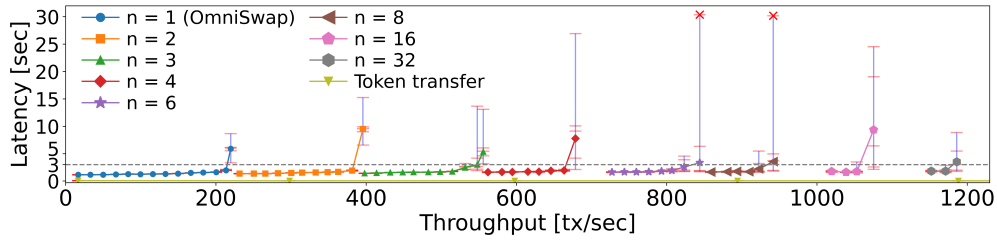
Since different protocols have distinct incentive structures, mechanism design must be tailored to each protocol. The non-parallelizable part of many popular smart contracts is less than 1 *transaction per second* (*tps*; see the full version [22]), well below the single-thread performance limit. Therefore, we focus on addressing the only urgent context we are aware of: *Automated Market Makers* (AMMs) [38, 39]. AMMs allow users to immediately swap token pairs by maintaining *liquidity pools*: tokens of both types supplied by other users serving as *liquidity providers*. It requires sequential handling of transactions since the outcome of each transaction depends on the current state of the token-pair pool and, in turn, alters this state.

We show that AMM performance does not scale in two state-of-the-art blockchain systems, Sui [14] and Solana [88]. The throughput is limited by a single thread at 214*tps* in Sui (Figure 1, $n = 1$) and 129*tps* in Solana (Figure 3). However, AMMs' demand is growing exponentially and by 2029 neither system could satisfy the most popular token pairs' trading demand (see the full version [22]). This encourages races and lengthens the queue for AMM trades, widening the time window in which adversaries can manipulate transaction ordering and thereby exacerbating front-running attacks such as sandwiching [94, 52]. To overcome this, users defer workload to expansion systems called layer-2 protocols [55, 2]; but this entails additional security assumptions.¹ Moreover, the single-thread bottleneck still exists in these layer-2 systems (e.g., 138*tps* in ammBoost [57]).

We apply sharding to AMMs to improve their throughput. We model the system (§3) as a set of AMM shards with rational users of two types: traders, who purchase tokens and seek to minimize expenses, and liquidity providers, who deposit tokens and earn fees.

The shards are AMMs based on the standard *Constant Product Market Maker* (CPMM) contract: Roughly, the contract maintains the product of the two tokens constant after each trade. Thus, purchasing a larger amount of a token increases its unit cost, an effect called *slippage*. See the full version [22] for more details.

¹ Blockchains are secured by financially-invested nodes. A layer-2 protocol relies on an additional set of nodes, with significantly less at stake [29], making it vulnerable to cheaper attacks.



■ **Figure 1** Sui trade latency as a function of demand with n SAMM shards.

We present SAMM (§4), an AMM protocol that uses multiple *shards* operating on the same blockchain, each with an independent liquidity pool. Ideally, all shards should be *balanced*, with the same liquidity; and traders should randomly select a shard to trade. Traders should not split trades across multiple shards, multiplying their overhead. Liquidity providers should not cause imbalances in shard sizes, making some shards less attractive to traders and reducing parallelism. Additionally, the system should re-stabilize if attackers unbalance the shards. Our goal is to disincentivize attacks that deviate from this ideal behavior, such as overloading the system by splitting trades across shards or concentrating trades in a subset of shards by unbalancing liquidity.

We prove that traditional CPMM shards are not secure. They charge a fixed-rate fee (e.g., 0.3% in Uniswap v2 [5]), which simply scales a trade’s cost by a constant factor relative to the no-fee trade. Traders are incentivized to route a single trade across multiple shards to reduce slippage.

At the core of SAMM is a novel *trading fee function*. SAMM generalizes the fixed-rate fee to a non-constant rate, incentivizing traders to use the smallest shard and avoid splitting trades. It is a simple monomial family, which is easy to parameterize and to analyze.

The model gives rise to a game (§5) played by the users: In each step, either a liquidity provider adds liquidity to a subset of the shards, or a trader executes a trade using a subset of the shards. We assume myopic liquidity-provider behavior, under which the liquidity provider treats the revenue from the next trade as an estimation for long-term revenue. This reduces the analysis to a Stackelberg game. Myopic utility is a common assumption in prior work [69, 34], and we further justify it in Section 5.1. Our analysis shows that, in all best responses, traders use one of the smallest shards. This implies that when shards are balanced, traders will randomly select a shard to trade, as intended. We also prove that shards converge (e.g., following intentional destabilization) to a balanced state in all Subgame-Perfect Nash Equilibria (SPNE) and present a specific SPNE, showing that SAMM is robust against irrational destabilization attacks. Malicious attackers suffer ongoing losses, and when their budget runs out, the system returns to a balanced state.

Before validating the theoretical analysis with a simulation, we turn our attention to the performance of SAMM, which will inform this simulation. We implement the protocol and deploy it in Sui and Solana local test networks (§6). This local validation is not an oversimplification: Our setting suffices to demonstrate the improvement in execution time, which is the bottleneck of these modern blockchains, as they process consensus separately from execution. We test *tps* and latencies of trades under different demand rates and numbers of shards. In Sui, the throughput increase is limited by the serial elements of Sui’s transaction processing, following Amdahl’s Law. In Solana, the throughput increases linearly with the number of shards up to a limit due to the blockchain’s constraints. SAMM achieves over a 5x throughput increase in Sui and a 16x increase in Solana, compared to a single-contract AMM.

Figure 1 shows that with more shards in Sui, SAMM achieves higher throughput (X axis) with low average latency (Y axis). Error bars show additional experiments, × marking failure due to overload. We observe similar results in Solana (see the full version [22]). Improving the blockchains’ parallelism would allow for even better performance.

The Mysten Labs Team reproduced our results and intends to make SAMM part of their benchmark suite.^{2,3} The RISE Chain benchmark demonstrates a 10x throughput improvement for multiple independent Uniswap instances. The team considers our design as necessary for scaling decentralized applications [67].³

Finally, we conduct a simulation to justify our theoretical assumptions (§7). We simulate traders that are free to split their trades using demand from real traces without synthetic arbitrage. Indeed, traders exhibit the behavior predicted by our theoretical analysis. Furthermore, simulation results quantify the economic impact of SAMM’s throughput gain. Splitting liquidity across shards worsens per-trade slippage, but each shard now processes more trades. As a result, a lower per-trade fee still yields substantially higher aggregate revenue for liquidity providers, and the lower fee offsets the added slippage, keeping traders’ costs comparable to a standard CPMM (and lower under some parameters). The same fee-slippage trade-off also lets SAMM absorb large trades (may split across shards), with smaller price impact than a single-pool CPMM. Hence, SAMM is attractive to both liquidity providers and traders when competing with other standard AMMs. Beyond the simulation, we also prove that, although split liquidity increases slippage, it neither worsens sandwich attacks nor harms liquidity providers when token prices fluctuate. Moreover, price fluctuations add only a negligible number of arbitrage transactions in the worst case.

In summary, our contributions are: (1) Identification of game-theoretic security challenges in sharded contracts, (2) quantification of the AMM performance bottleneck, (3) SAMM: a sharded AMM contract, (4) generalization of the AMM trading fee function and a novel function for SAMM, (5) game-theoretic analysis identifying a Subgame-Perfect Nash Equilibrium (SPNE), (6) evaluation demonstrating an increase in throughput of 5x in Sui and of 16x in Solana (up to each blockchain’s limit), (7) simulation with real trade data confirming the effects of SAMM’s incentive design, and (8) validation that sharding does not incentivize sandwich attacks, and that it neither increases losses nor incurs substantial transaction overhead under price fluctuations.

These results provide an initial demonstration that high-demand smart contracts can be scaled securely by sharding a single logical service into multiple identical instances (§8). They also hint at an upcoming challenge in smart-contract platform design: minimizing the serial elements of transaction processing. But SAMM can already be employed to scale AMM performance.

2 Related Work

Addressing blockchain throughput limitations has been a central research focus since its inception. The first generation of blockchains, starting with Bitcoin [63], suffered from throughput limitations due to their consensus protocols. A body of work overcame this with a variety of protocols [33, 91, 88, 73, 28, 10, 1, 41, 51, 68, 89, 46, 82]. With consensus constraints out of the way, the serial execution of transactions became the bottleneck.

² From personal communication.

³ The authors are not affiliated with Mysten Labs, Solana, or RISE and have no financial interest in their tokens.

Subsequent approaches identify read/write set conflicts and parallelize non-conflicting smart contract transaction execution [72, 30, 54, 65, 37]. Numerous practical blockchain platforms adopt such parallel execution engines: some require applications to explicitly shard independent components to expose parallelism [14, 88], while others automatically parallelize non-conflicting transactions [49]. On the former type of platforms, existing smart contracts (stateful programs) often rely on naive sharding, which partitions an independent component into multiple instances [61, 45, 17, 44], to benefit from parallel execution. However, non-parallelizable operations within a smart contract remain a bottleneck. To the best of our knowledge, no prior work has addressed improving the throughput of non-parallelizable smart contracts via sharding and studied the associated security challenges.

Uniswap v1 and v2 [92, 5] established the *Constant Product Market Maker* (CPMM) as a widely adopted, order-book-free design for token exchange [76, 64]. Uniswap v3 [6] and Curve [26] further improved AMMs' ability to handle larger trades. Uniswap v4 [9] and UniswapX [3] introduced practical updates to how pools are organized and trades are executed. Our sharding design targets parallelism and incentive alignment and can be composed with them.

Research on AMMs has studied generalized market-making mechanisms [7], *routers* which maximize trading utility by using multiple AMMs [8, 32, 93], advanced arbitrage techniques [94, 47, 11, 81, 83], improving liquidity providers' returns [39], measuring loss due to arbitrage [59, 36, 58], ensuring transaction privacy [24], eliminating Miner Extractable Value (MEV) for fair trades [20, 85, 18, 42], and examining the synergy between blockchain-based AMMs and prediction market mechanisms [70]. They do not consider AMM throughput and its security implications.

AMM throughput is limited by a single thread in current blockchains. Some AMM contracts [48, 66, 57, 4, 79] are deployed on so-called layer-2 solutions [2, 40, 12, 15] to defer some workload off-chain, but this merely creates a separate environment for AMMs, with scaling issues persisting (e.g., 138tps in ammBoost [57]) within this realm while introducing additional security assumptions.¹ Some existing AMMs use multiple instances of a single token pair with different parameters for flexibility [6], but most of the demand concentrates on a single instance [50]. To the best of our knowledge, previous work did not address the single-thread bottleneck of smart contracts in general and specifically for AMMs.

3 Model

We abstract away the blockchain details for our analysis and model the system as a set of participants interacting directly with AMMs, exchanging tokens. The system progresses in discrete steps and there exists an external market used by arbitrageurs to arbitrage in our system. The model uses a generic AMM, which we will later instantiate based on previous work and with SAMM.

Participants. There are two types of participants, *liquidity providers* and *traders*. Each liquidity provider holds some *token A* and *token B*. They aim to increase their holdings. Traders are either *AB* or *BA*, based on their goals. Each *BA* trader occasionally wishes to get a certain amount of *token A*, and vice versa for *AB* traders. They have sufficiently many tokens of the opposite type to complete their trade, but they aim to minimize the cost of obtaining the desired tokens. Both liquidity providers and traders can send and receive tokens to and from the smart contracts.

Automated Market Makers. The system also includes Automated Market Makers, stateful programs that facilitate *depositing* and *trading* of tokens. Each AMM maintains some deposited amounts R^A of *token A* and R^B of *token B*. We call these tokens *liquidity*.

Within an AMM, there are three primary operations: *liquidity addition*, *liquidity removal*, and *trade*. Specifically, a liquidity provider deposits I^A *token A* and I^B *token B* to the contract (liquidity addition); withdraws O^A *token A* and O^B *token B* from the contract (liquidity removal); or a trader sends I^A *token A* (resp., I^B *token B*) to the contract and gets O^B *token B* (resp., O^A *token A*) from the contract (trade).

In a trade operation, the required input amount for a trader to receive a specific output amount of another token depends on both the output amount and the AMM's current state. We define the gross amount of a *BA* trade as the required input amount of *token B* to get O^A *token A* in the AMM. We denote it by $G(R^A, R^B, O^A)$ (resp., $G(R^B, R^A, O^B)$ for *AB* traders). Within the gross amount, traders pay a so-called *trading fee* which contributes to the liquidity providers' revenue.

Liquidity providers supply liquidity by depositing their tokens in the contract. Once contributing to the AMM, a provider receives tokens from trading fees, hence earning revenue. Liquidity providers can later withdraw their tokens from the AMM.

Following prior work [59, 39], we assume that the trading fee is directly paid to the liquidity providers. Although some practical AMMs (e.g., Uniswap v2 [5]) reinvest the trading fees into themselves, allowing liquidity providers to withdraw more tokens than they deposited as utilities, the trading fee's impact is negligible relative to the deposited amount: The average ratio of the output amount to the deposited amount is very small (less than 0.036% as we find in the full version [22]), and automated market makers (AMMs) typically charge a low trading fee relative to the gross amount (e.g., 0.3% in Uniswap), so the trading fee's impact is negligible relative to the amount of deposited tokens.

System State and Progress. There are n independent AMM shards $shard_1, shard_2, \dots, shard_n$ in the system. The system progresses in discrete steps $k = 0, 1, 2, \dots$ and is orchestrated by a *scheduler*. In each step, the scheduler randomly selects a participant and this participant executes transactions. It chooses a liquidity provider with probability $P_{lp} \geq 0$ and a trader with probability $P_t \geq 0$, where $P_{lp} + P_t = 1$. The scheduler assigns the liquidity provider l^A *token A* and l^B *token B*, where (l^A, l^B) follows a random distribution D_{lp} .

The trader is either a *BA* trader aiming to obtain *token A* or an *AB* trader aiming to obtain *token B*. The probability of drawing an *AB* trader (resp., *BA* trader) is $P_t^{AB} \geq 0$ (resp., $P_t^{BA} \geq 0$), with $P_t^{AB} + P_t^{BA} = P_t$. To avoid repetition, we only show the case of *BA* traders; the expressions for *AB* traders are symmetric. The scheduler assigns the *BA* trader an amount of b^{BA} *token A* to obtain, following a random distribution D^{BA} .

External Market and Arbitrageurs. Following prior work [39, 59], we assume there is an *external market* providing the price of *token A* and *token B*, p^A and p^B , respectively. These prices do not change due to trades; they serve as objective prices for *token A* and *token B*.

If the AMM sells tokens at a price lower than the external market, a principal can buy tokens from the AMM and sell them in the external market to make profits, and vice versa; this is called an *arbitrage*. Previous work [59, 39, 18, 20] assumes active and rational arbitrageurs who can use the external market and arbitrage to maximize their utility. We follow the assumption [59] that there are immediate arbitrages without trading fees when the token price in the AMM is different from the external market.

Threat Model and Our Goal. As is typical in blockchain security research [90, 43, 60, 19, 87, 78, 56, 27], rational participants are potential attackers that might deviate from the intended protocol to maximize their own utility. In our context, attackers might, for example, overload the system or create temporary imbalances that reduce parallelism. Our goal is to design a set of AMM shards such that all rational participants behave in a way that improves the overall throughput of the system.

4 SAMM: Sharded AMM

We present SAMM, an AMM comprising multiple shards. Each shard is an independent AMM (§4.1), and SAMM’s goal is to have operations distributed among the different shards (§4.2). To enforce this desired behavior despite attacks and untrusted users, we present a novel trading fee function (§4.3) and find parameters to fulfill SAMM’s goal (§4.4).

4.1 SAMM Structure

SAMM enables parallel processing by deploying multiple AMM shards, each with an independent liquidity pool. There is no data dependency among the shards and there are no global elements. The operation of each shard is based on the Constant Product Market Maker (See detailed overview in the full version [22]), which keeps the product of the two tokens constant.

When liquidity providers add tokens to a shard, they receive *share tokens*, which signify their portion of its liquidity. Assume the shard has R^A token A and R^B token B deposited, and R^S is the number of all share tokens distributed before the operation. The amount of share tokens acquired by the liquidity provider is $O^S = R^S \times \min \left\{ \frac{I^A}{R^A}, \frac{I^B}{R^B} \right\}$. If a liquidity provider pays I^S share tokens to withdraw deposited tokens, she gets $O^A = \frac{I^S}{R^S} \times R^A$ token A and $O^B = \frac{I^S}{R^S} \times R^B$ token B .

When a trader wants to get O^A token A from the AMM, ignoring trading fees, she needs to pay the *net amount*

$$\text{net}(R^A, R^B, O^A) = \frac{R^B \times O^A}{R^A - O^A} . \quad (1)$$

The *net price* for a single token A is $\frac{R^B}{R^A - O^A}$ token B . This value increases as the output amount of token A , O^A , increases, which is the *Slippage* of the trade.

4.2 Desired Properties

Most operations in an AMM are trades (We observe 99.5% in historical blockchain records, see the full version [22]). Our main goal is therefore that traders distribute the workload evenly among the shards. The following properties will suffice.

c-Non-Splitting Property. Our first goal is that trades only use a single shard, i.e., traders do not split a trade into smaller ones on multiple shards. This avoids the overhead caused by trade splits. So, the cost of a single-trade transaction should be less than the combined cost of multiple, split-trade transactions. However, this principle faces a significant challenge as highlighted by Equation 1: When the output amount is not small enough in comparison to the shard’s deposited amount, the resulting slippage could incentivize traders to split their transactions to mitigate this slippage. Consequently, we refine our requirement to ensure that

this principle is adhered to only when the output amount is relatively small compared to the deposited amount, specifically when the ratio is below a predefined constant, c . Indeed, the prominent pairs in Ethereum’s Uniswap v2 data support this approach, with 99% of trades having a ratio of output amount to deposited amount below 0.0052 (See the full version [22] for more details). We call this the *c-non-splitting* property.

► **Property 4.1** (*c-non-splitting*). *Let $m \geq 2$. Given a set of m output amounts $\{O_1^A, O_2^A, \dots, O_m^A\}$ such that all amounts are positive, i.e., $\forall 1 \leq j \leq m : O_j^A > 0$, denote by $\tilde{O}^A$ the sum of the amounts in the set, $\tilde{O}^A = \sum_{j=1}^m O_j^A$. For the constant $0 < c < 1$ and the deposited amount of tokens R^A and R^B , if $\frac{\tilde{O}^A}{R^A} \leq c$, then the cost of trading $\tilde{O}^A$ token A is less than the sum of the cost of trading O_j^A token A for $1 \leq j \leq m$, i.e., $G_{\text{SAMM}}(R^A, R^B, \tilde{O}^A) < \sum_{j=1}^m G_{\text{SAMM}}(R^A, R^B, O_j^A)$.*

c-Smaller-Better Property. Our second goal is to maintain balanced volumes in all shards. This is crucial because the volume affects the slippage. When there are differences in shard sizes, with some being much smaller than others, the slippage in trading within these smaller shards is significantly greater than in larger ones. This discrepancy can result in transactions clustering in the larger shards rather than spread evenly, reducing parallelism. Moreover, an attacker could disrupt the system by adding or removing liquidity from certain shards, thereby harming overall system performance.

We address this by incentivizing liquidity providers to allocate their tokens to the shards with lower volumes. Intuitively, this ensures that smaller shards receive more frequent fees from traders when shard volumes are not balanced. This incentivizes liquidity providers to deposit tokens in these smaller shards. Therefore, the system would converge to the state where all shards have balanced volumes, and traders then randomly select shards for trading. As with the *c-non-splitting* property, large transactions suffer from high slippage, resulting in an advantage for larger shards. So here too, we refine this requirement to scenarios where the ratio of the traded amount to the deposited amount is below a threshold c . We call this the *c-smaller-better* property.

► **Property 4.2** (*c-smaller-better*). *Given an output amount $O^A > 0$, for any two shards with deposited token amounts (R_i^A, R_i^B) and (R_j^A, R_j^B) , respectively, and $R_i^A < R_j^A$. For the constant $0 < c < 1$, if $O^A/R_j^A < O^A/R_i^A \leq c$ and $R_i^A/R_i^B = R_j^A/R_j^B$, then the cost of trading O^A token A in the smaller shard is less than that in the larger shard, i.e., $G_{\text{SAMM}}(R_i^A, R_i^B, O^A) < G_{\text{SAMM}}(R_j^A, R_j^B, O^A)$.*

c value. If the above properties are satisfied for a particular c but a trade occurs with a larger ratio of output amount to the deposited amount, traders may split their transactions or tend to use larger shards to minimize their cost. Therefore, c should be as large as possible to ensure such occurrences are rare.

Note. CPMs charge a constant ratio of the net amount as the trading fee. Therefore, their cost function does not satisfy either property due to slippage (see the full version [22]).

4.3 Trading Fee Design

To satisfy properties 4.1 and 4.2, we first generalize the trading fee function to provide flexibility in the incentive design. Then, we propose a specific trading fee function.

Generic Trading Fee Function. The gross amount of a trade comprises the trading fee and the net amount, with the latter being determined by the CPMM curve. To maintain the foundational characteristics of AMMs, we do not modify the CPMM curve. Instead, we generalize the trading fee function beyond simply charging a ratio of the net amount as in previous work (e.g., [7, 39, 8]).

Denote the trading fee function of the AMM by $tf(R^A, R^B, O^A)$, which takes the deposited amount of *token A*, R^A , and *token B*, R^B , in the shard and the output amount of *token A*, O^A , and outputs the amount of *token B* the trader needs to pay as the trading fee. Then, the gross amount of getting O^A *token A* is:

$$G(R^A, R^B, O^A) = tf(R^A, R^B, O^A) + net(R^A, R^B, O^A). \quad (2)$$

Bounded-Ratio Trading Fee Function. In order to achieve the desired properties, we need flexibility for the trading fee function design. A monomial function is easy to parameterize and to differentiate, and is sufficient to achieve most of our goals. The function takes the variables available on a trade, R^A, R^B, O^A . It is parameterized by four values, $\beta_1, \beta_2, \beta_3, \beta_4$: $tf(R^A, R^B, O^A; \beta_1, \beta_2, \beta_3, \beta_4) := \beta_1(R^A)^{\beta_2}(R^B)^{\beta_3}(O^A)^{\beta_4}$.

The monomial is unbounded: Without limits, the trading fee can be too high to attract traders or too low to reward liquidity providers. Analogously to the fixed ratio in prior work, we thus add adjustable bounds $r_{\min}, r_{\max}$ and a base offset β_5 to fine-tune the fee's absolute value, yielding the bounded-ratio polynomial:

$$tf_{BRP}(R^A, R^B, O^A; \beta_1, \beta_2, \beta_3, \beta_4, \beta_5) := \frac{R^B}{R^A} O^A \times \max\{r_{\min}, \min\{r_{\max}, \beta_1(R^A)^{\beta_2}(R^B)^{\beta_3}(O^A)^{\beta_4} + \beta_5\}\}. \quad (3)$$

The ratio R^B/R^A represents the market price of *token A* relative to *token B* (see the full version [22]). The product $\frac{R^B}{R^A} O^A$ is thus the trader's net payment in terms of *token B* without slippage. Then $\max\{r_{\min}, \min\{r_{\max}, \beta_1(R^A)^{\beta_2}(R^B)^{\beta_3}(O^A)^{\beta_4} + \beta_5\}\}$ acts as the constrained ratio of the trading fee to that net payment. We hereinafter omit the parameters and write $tf_{BRP}(R^A, R^B, O^A)$ for brevity.

4.4 Parameter Selection

Now we turn to the selection of parameters for the SAMM trading fee function. First, we identify the necessary conditions under which the bounded-ratio polynomial trading fee function aligns with the *c*-smaller-better property.

► **Proposition 4.3.** *Let $tf_{SAMM}(R^A, R^B, O^A) = tf_{BRP}(R^A, R^B, O^A)$, then the following conditions are necessary for *c*-smaller-better to hold for $G_{SAMM}(R^A, R^B, O^A)$: (1) $\beta_3 = 0$; (2) $\beta_2 + \beta_4 = 0$; (3) $\beta_1 < 0$; (4) $0 < \beta_4 \leq 1$; (5) $r_{\min} < \beta_5 \leq r_{\max}$; and (6) $\frac{\beta_5 - r_{\min}}{-\beta_1} \geq c^{\beta_4}$.*

We require the polynomial value to be between $r_{\min}$ and $r_{\max}$. Then, we need to make sure that the derivative of the gross amount with respect to the shard size is non-negative to ensure the *c*-smaller-better property. Required items come directly from these two restrictions. The proof is in the full version [22].

Based on Proposition 4.3, we require that $\beta_1 < 0$, $\beta_2 + \beta_4 = 0$, $\beta_3 = 0$, $0 < \beta_4 \leq 1$, and $r_{\min} < \beta_5 \leq r_{\max}$. Next, we identify additional conditions that are sufficient for both properties to hold.

► **Theorem 4.4.** *Let $tf_{SAMM}(R^A, R^B, O^A) = tf_{BRP}(R^A, R^B, O^A)$, if $\beta_1 < 0, \beta_2 + \beta_4 = 0, \beta_3 = 0, 0 < \beta_4 \leq 1, r_{\min} < \beta_5 \leq r_{\max}$ and $\frac{\beta_5 - r_{\min}}{-\beta_1} \geq c^{\beta_4}$, then the following items are sufficient for the c -non-splitting and c -smaller-better properties to hold for $G_{SAMM}(R^A, R^B, O^A)$: (1) $\beta_1\beta_4(\beta_4 + 1)c^{\beta_4-1}(1 - c)^3 \leq -2$; and (2) $-\beta_1\beta_4 \geq \frac{c^{1-\beta_4}}{(1-c)^2}$.*

The c -smaller-better property is satisfied when the derivative of the gross amount is positive. It is sufficient to ensure the c -non-splitting property when the gross amount is concave with respect to the output amount, which is ensured by a negative second derivative over the output amount. The proof is in the full version [22].

By setting $\beta_2 = -1, \beta_3 = 0, \beta_4 = 1, \beta_5 = r_{\max}$, and choosing $\beta_1 < -1$, the fee function satisfies the above requirements, leaving just three parameters:

$$tf_{SAMM}(R^A, R^B, O^A) = \frac{R^B}{R^A} \times O^A \times \max \left\{ r_{\min}, \beta_1 \times \frac{O^A}{R^A} + r_{\max} \right\}.$$

By setting $\beta_4 = 1$ and $\beta_5 = r_{\max}$ in the sufficient condition of the c -non-splitting and c -smaller-better properties (Theorem 4.4), the sufficient condition becomes:

► **Corollary 4.5.** *For any $\beta_1 < -1$ and c satisfying $c \leq \min \left\{ 1 - (-\beta_1)^{-\frac{1}{3}}, \frac{r_{\max} - r_{\min}}{-\beta_1} \right\}$, the SAMM cost function $G_{SAMM}(R^A, R^B, O^A)$ satisfies the c -non-splitting and c -smaller-better properties.*

For instance, the parameters $\beta_1 = -1.05, r_{\max} = 0.012, r_{\min} = 0.001$, and $c = 0.0104$ meet the specified criteria. According to historical records (see the full version [22]), over 99% of Uniswap v2 transactions have a ratio below 0.0052, suggesting that if its liquidity is split into two shards, 99% of transactions would fall within our targeted range. Specifically, in the AMMs with the highest trading volumes, USDC-ETH and USDT-ETH, the ratio remains below 0.00128, which can manage eight shards. Increasing the number of shards could be achieved by adjusting the value of c by increasing $r_{\max}$ and β_1 .

5 Analysis of Game-Theoretic Security

The model gives rise to a game (§5.1) played by traders and liquidity providers. Our goal (§5.2) is to ensure that at equilibrium, traders randomly select shards to trade in without splitting, thereby enhancing throughput. SAMM achieves this (§5.3) and converges to a state where all shards have balanced volumes, overcoming attacks that unbalance the shards.

5.1 Game Model

We derive from the model a sequential game with discrete steps $k = 0, 1, \dots$. Denote the game, parameterized by the number n of shards and their trading fee function tf , by $\Gamma_n(tf)$.

System State. In $\Gamma_n(tf)$, the state of each shard $shard_i$ in step k consists of the amount of deposited *token A*, the amount of deposited *token B*, and the amount of share tokens, $R_i^A(k), R_i^B(k), R_i^S(k)$, respectively. Recall that share tokens are not deposited in the shard but are held by liquidity providers, so R_i^S is the total amount of $shard_i$'s share tokens held by liquidity providers. Denote by $\mathbf{R}(k)$ the state of all AMM contracts in step k , $((R_1^A(k), R_1^B(k), R_1^S(k)), \dots, (R_n^A(k), R_n^B(k), R_n^S(k)))$.

Liquidity Provider Actions. The liquidity provider decides the amount of tokens she deposits in each shard. Denote the amount of *token A* and *token B* she deposits in *shard_i* by $l_i^A, l_i^B \geq 0$, respectively. Recall that the scheduler assigns the liquidity provider l^A *token A* and l^B *token B*. The total amount of tokens deposited should not exceed the amount she holds:

$$\forall 1 \leq i \leq n, l_i^A, l_i^B \geq 0, \sum_{i=1}^n l_i^A \leq l^A, \sum_{i=1}^n l_i^B \leq l^B.$$

The action of a liquidity provider is thus the vector $a_{lp} = ((l_1^A, l_1^B), \dots, (l_n^A, l_n^B))$. When the liquidity provider takes this action with the system state $\mathbf{R}(k)$, the liquidity provider receives O_i^S share tokens from *shard_i*, where $O_i^S = R_i^S(k) \times \min \left\{ \frac{l_i^A}{R_i^A(k)}, \frac{l_i^B}{R_i^B(k)} \right\}$.

There is no arbitrage opportunity only if $R_i^A/R_i^B = p^B/p^A$ (see the full version [22]), so arbitrageurs enforce this equality. When $l_i^A/l_i^B = p^B/p^A$, increasing just one of l_i^A or l_i^B would not increase the share token the liquidity provider receives, which means more payment without more revenue. Therefore, we only consider actions where $l_i^A/l_i^B = p^B/p^A$ and require $l^A/l^B = p^B/p^A$. The action space of a liquidity provider is denoted by $\mathcal{A}_{lp}(l^A, l^B)$:

$$\mathcal{A}_{lp}(l^A, l^B) = \left\{ a_{lp} \left| \begin{array}{l} \forall 1 \leq i \leq n, l_i^A = \frac{p^B}{p^A} l_i^B \geq 0, \\ \sum_{i=1}^n l_i^A \leq l^A, \sum_{i=1}^n l_i^B \leq l^B \end{array} \right. \right\}.$$

We denote the updated state of shards from the previous state $\mathbf{R}$ and the action of a liquidity provider a_{lp} by $\mathbf{R} + a_{lp}$. Then for $\mathbf{R}' = ((R_1^{A'}, R_1^{B'}, R_1^{S'}), \dots, (R_n^{A'}, R_n^{B'}, R_n^{S'})) = \mathbf{R} + a_{lp}$, we have

$$R_i^{A'} = R_i^A + l_i^A, R_i^{B'} = R_i^B + l_i^B, R_i^{S'} = R_i^S + O_i^S = \left(1 + \frac{l_i^A}{R_i^A}\right) R_i^S.$$

After the liquidity addition operation, there is no arbitrage opportunity for the arbitrageurs since $\frac{R_i^{A'} + l_i^A}{R_i^{B'} + l_i^B} = \frac{R_i^A}{R_i^B} = \frac{p^B}{p^A}$ (see the full version [22]). Then the update of state in step $k+1$ is $\mathbf{R}(k+1) = \mathbf{R}(k) + a_{lp}$.

Trader Actions. The action of a *BA* trader determines the amount of *token A* she acquires from each shard. Denote by $b_i^{BA} \geq 0$ the amount of *token A* she acquires in *shard_i*. Recall that the scheduler assigns the *BA* trader b^{BA} *token A* to acquire in total. The action of a *BA* trader is thus the vector $a^{BA} = (b_1^{BA}, \dots, b_n^{BA})$. The action space of a *BA* trader is denoted by $\mathcal{A}^{BA}(b^{BA})$, which is the set of all feasible actions:

$$\mathcal{A}^{BA}(b^{BA}) = \left\{ a^{BA} \left| \forall 1 \leq i \leq n, b_i^{BA} \geq 0, \sum_{i=1}^n b_i^{BA} = b^{BA} \right. \right\}. \quad (4)$$

After the trade operation and the arbitrage, R_i^A and R_i^B remain unchanged (see the full version [22]). Consequently, the state of the shards remains unchanged in the subsequent step: $\forall 1 \leq i \leq n, R_i^A(k+1) = R_i^A(k), R_i^B(k+1) = R_i^B(k)$.

Utility and Strategies. For traders and liquidity providers, we first discuss their revenue and then define their strategies and utility, respectively. Players determine the value of tokens according to the external market, namely p^A and p^B .

Consider a BA trader whose goal is to acquire b^{BA} units of *token A*. This trader needs to pay the gross amount and may derive some fixed reward from getting these tokens. We consider her revenue only as the inverse of the gross amount in terms of *token B* times the value of each *token B*:

$$U^{BA}(\mathbf{R}, a^{BA}) = -p^B \times \sum_i G(R_i^A, R_i^B, b_i^{BA}) = -p^B \times \sum_i G(R_i^A, \frac{p^A}{p^B} R_i^A, b_i^{BA}) . \quad (5)$$

For the trader aiming to get b^{BA} *token A*, the strategy of the trader $\pi^{BA}(\mathbf{R}, b^{BA}, a^{BA})$ takes $\mathbf{R}$, b^{BA} and an action a^{BA} as input, then outputs the probability of taking action a^{BA} . The total probability of all feasible actions should be 1: $\sum_{a^{BA} \in \mathcal{A}^{BA}(b^{BA})} \pi^{BA}(\mathbf{R}, b^{BA}, a^{BA}) = 1$.

The utility of the trader over the strategy is a function of the system state $\mathbf{R}$, the assigned requirement b^{BA} , and the strategy of the trader π^{BA} . It is the expected revenue under the distribution of actions:

$$U^{BA}(\mathbf{R}, b^{BA}, \pi^{BA}) = \sum_{a^{BA} \in \mathcal{A}^{BA}(b^{BA})} (\pi^{BA}(\mathbf{R}, b^{BA}, a^{BA}) \times U^{BA}(\mathbf{R}, a^{BA})) . \quad (6)$$

The revenue of a liquidity provider comes from the trading fees paid by traders. Ignoring the effect of other liquidity providers, all future steps are identical due to arbitrageurs, so the long-term mean revenue is proportional to the next-step mean revenue. Hence, we consider the *myopic* setting (as in, e.g., [69, 34]), where the liquidity provider regards the revenue in the next step as her utility.

Denote the revenue of a liquidity provider with her action $a_{lp} = ((l_1^A, l_1^B), \dots, (l_n^A, l_n^B))$, the action of the BA trader in the next step $a^{BA} = (b_1^{BA}, \dots, b_n^{BA})$ and the system state $\mathbf{R} = ((R_1^A, R_1^B, R_1^S), \dots, (R_n^A, R_n^B, R_n^S))$, by the function $U_{lp}(\mathbf{R}, a_{lp}, a^{BA})$. In the next step, $shard_i$ receives a trading fee of $tf(R_i^A + l_i^A, R_i^B + l_i^B, b_i^{BA})$. The liquidity provider receives a fraction of that fee proportional to her fraction of share tokens out of all shares in the shard. Therefore, the revenue function is

$$\begin{aligned} U_{lp}(\mathbf{R}, a_{lp}, a^{BA}) &= p^B \times \sum_{i=1}^n \left\{ tf(R_i^A + l_i^A, R_i^B + l_i^B, b_i^{BA}) \times \frac{\frac{l_i^A}{R_i^A} R_i^S}{\left(1 + \frac{l_i^A}{R_i^A}\right) R_i^S} \right\} \\ &= p^B \times \sum_{i=1}^n \left\{ tf(R_i^A + l_i^A, R_i^B + l_i^B, b_i^{BA}) \times \frac{l_i^A}{l_i^A + R_i^A} \right\} . \end{aligned} \quad (7)$$

For the liquidity provider with l^A *token A* and l^B *token B*, the strategy of the liquidity provider $\pi_{lp}(\mathbf{R}, l^A, l^B)$ takes $\mathbf{R}$, l^A , l^B and an action a_{lp} as input, and outputs the probability of taking action a_{lp} . The total probability of all feasible actions should be 1,

$$\sum_{a_{lp} \in \mathcal{A}_{lp}(l^A, l^B)} \pi_{lp}(\mathbf{R}, l^A, l^B, a_{lp}) = 1 . \quad (8)$$

The utility of the liquidity provider over strategies is a function of the system state $\mathbf{R}$, the amount of tokens she is assigned l^A , l^B , and the strategy of the liquidity provider and traders, $\pi_{lp}, \pi^{BA}, \pi^{AB}$; denote it by $U_{lp}(\mathbf{R}, l^A, l^B, \pi_{lp}, \pi^{BA}, \pi^{AB})$. Before calculating this, we show the revenue given the action of the liquidity provider and the strategies of traders, denoted by $U_{lp}(\mathbf{R}, l^A, l^B, a_{lp}, \pi^{BA}, \pi^{AB})$. It takes the system state $\mathbf{R}$, the action of the liquidity provider a_{lp} , the strategies of traders π^{BA} and π^{AB} as input, then outputs the expected utility over the strategies and distributions of traders. The strategy of traders is affected by the state after the liquidity provider's action, namely $\mathbf{R} + a_{lp}$. Denote by $E_{b^{BA} \sim D^{BA}}[f(\cdot)]$ the expected value of $f(\cdot)$ with b^{BA} sampled from D^{BA} . The revenue is:

$$\begin{aligned}
U_{lp}(\mathbf{R}, l^A, l^B, a_{lp}, \pi^{BA}, \pi^{AB}) &= P_t^{BA} \times E_{b^{BA} \sim D^{BA}} \left[\sum_{a^{BA} \in \mathcal{A}^{BA}(b^{BA})} \left(\frac{\pi^{BA}(\mathbf{R} + a_{lp}, b^{BA}, a^{BA})}{U_{lp}(\mathbf{R}, a_{lp}, a^{BA})} \times \right) \right] \\
&+ P_t^{AB} \times E_{b^{AB} \sim D^{AB}} \left[\sum_{a^{AB} \in \mathcal{A}^{AB}(b^{AB})} \left(\frac{\pi^{AB}(\mathbf{R} + a_{lp}, b^{AB}, a^{AB})}{U_{lp}(\mathbf{R}, a_{lp}, a^{AB})} \times \right) \right].
\end{aligned}$$

To simplify the presentation, we assume the liquidity provider is always followed by a BA trader. The expressions for an AB trader are symmetric. Then, the above equation can be simplified as

$$U_{lp}(\mathbf{R}, l^A, l^B, a_{lp}, \pi^{BA}) = E_{b^{BA} \sim D^{BA}} \left[\sum_{a^{BA} \in \mathcal{A}^{BA}(b^{BA})} \left(\frac{\pi^{BA}(\mathbf{R} + a_{lp}, b^{BA}, a^{BA})}{U_{lp}(\mathbf{R}, a_{lp}, a^{BA})} \times \right) \right]. \quad (9)$$

Then, the utility function of the liquidity provider over strategies is the expected utility under the distribution of actions:

$$U_{lp}(\mathbf{R}, l^A, l^B, \pi_{lp}, \pi^{BA}, \pi^{AB}) = \sum_{a_{lp} \in \mathcal{A}_{lp}(l^A, l^B)} \left(\frac{\pi_{lp}(\mathbf{R}, l^A, l^B, a_{lp})}{\times U_{lp}(\mathbf{R}, l^A, l^B, a_{lp}, \pi^{BA}, \pi^{AB})} \right). \quad (10)$$

Solution Concept. In a Subgame-Perfect Nash Equilibrium (SPNE), players cannot gain higher utility by changing strategies at any step [71], knowing that subsequent players will take their best responses.

When a trader takes an action in a given step, her utility is influenced solely by her immediate strategy and the current state of AMMs, as outlined in Equation 5. Crucially, future actions do not affect this calculation, allowing the trader to directly optimize her utility, thereby establishing dominant strategies.

In the case of a liquidity provider being chosen in a step, the situation is different. Given their myopic viewpoint, liquidity providers only need to account for the strategy of the trader in the ensuing step. Their actions in subsequent steps do not affect their own utility. Thus, the sequential game is reduced to a two-stage Stackelberg game and SPNE to a Stackelberg Equilibrium [80].

To formalize this, we denote the strategies of the liquidity provider, the BA trader, and the AB trader in the SPNE by τ_{lp} , τ^{BA} and τ^{AB} , respectively. The BA trader would always get the optimal utility in equilibrium, namely $\forall \mathbf{R}, b^{BA}$, we have $U^{BA}(\tau^{BA}, \mathbf{R}, b^{BA}) = \max_{\pi^{BA}} U^{BA}(\pi^{BA}, \mathbf{R}, b^{BA})$.

Note that τ^{BA} is a best response for the BA trader. The strategy of liquidity provider in equilibrium is just the optimal strategy when traders adopt their best response, namely for all $\mathbf{R}, l^A$ and l^B , we have $U_{lp}(\tau_{lp}, \mathbf{R}, \tau^{BA}, \tau^{AB}, l^A, l^B) = \max_{\pi_{lp}} U_{lp}(\pi_{lp}, \mathbf{R}, \tau^{BA}, \tau^{AB}, l^A, l^B)$.

Game Assumptions. We briefly discuss three non-trivial model assumptions.

First, we did not explicitly consider liquidity removal in the game. If a liquidity provider removes all her liquidity, it just changes the state of the AMM pools and our analysis still holds. If the liquidity provider strategically removes liquidity, it is equivalent to removing all liquidity first and then adding liquidity again. Therefore, we do not lose generality by ignoring liquidity removals.

Second, in a blockchain, each transaction consumes resources measured in a metric called *gas*. The transaction pays a fee according to its gas consumption. In practice, gas fees are volatile, often very low in high-performance blockchains (e.g., in Sui under \$0.005

per transaction on average [75, 77]). The model conservatively neglects gas fees to avoid arbitrary assumptions on gas costs. Gas fees disincentivize trade splitting, which helps with parallelization. In particular, with a set gas fee, Properties 4.1 and 4.2 continue to hold. Therefore, it can only strengthen the equilibrium analysis.

Third, perfect arbitrageurs are a common assumption in the literature [59, 39, 18, 20]. We nonetheless demonstrate later (§7) that our results hold with real workloads, without this assumption. In addition, we do not model arbitrage that trades across SAMM shards or between SAMM and other decentralized exchanges (DEXs). We call this type of arbitrage *DEX-to-DEX*. Any DEX-to-DEX arbitrage can be replaced by two independent trades, each between a DEX and the external market, yielding the same payoff. Therefore, DEX-to-DEX arbitrage is weakly dominated by arbitrages with the external market. Empirically, DEX-to-DEX arbitrage accounts for less than 10% of observed arbitrage activity [25].

5.2 Desired Property

Our goal is to improve the throughput by allowing parallelism. Specifically, we would like traders to evenly distribute their transactions among all AMM shards without splitting them. That is, a dominant strategy for the BA trader should be to randomly select an AMM shard to acquire all her needed *token A*. Denote the action of getting all b^{BA} *token A* in *shard_i* by

$$a_i^{BA}(b^{BA}) = (0, \dots, b_i^{BA} = b^{BA}, \dots, 0) . \quad (11)$$

Denote the set of these actions by $\mathcal{A}_1(b^{BA}) = \{a_i^{BA}(b^{BA}) | 1 \leq i \leq n\} \subset \mathcal{A}_t^{BA}(b^{BA})$. The strategy that uniformly at random selects an AMM contract to acquire all her needed *token A* is the *perfect parallelism strategy*:

► **Definition 5.1.** *The perfect parallelism strategy of the BA trader is*

$$\hat{\tau}^{BA}(\mathbf{R}, b^{BA}, a^{BA}) = \begin{cases} \frac{1}{n}, & \text{if } a^{BA} \in \mathcal{A}_1(b^{BA}) \\ 0, & \text{otherwise} \end{cases} .$$

Our goal is thus to have the perfect parallelism strategy be a dominant strategy:

► **Property 5.2.** *The perfect parallelism strategy of the BA trader is a dominant strategy:*

$$\forall \pi^{BA} : U^{BA}(\pi^{BA}, \mathbf{R}, b^{BA}) \leq U^{BA}(\hat{\tau}^{BA}, \mathbf{R}, b^{BA}) .$$

Using multiple CPMs does not satisfy the perfect parallelism property and would be counterproductive: Each trader would split her transactions across all AMM contracts. Thus, although the total number of trades increases due to parallelism, the satisfied trade demand is no higher than a single AMM contract and possibly lower since the total throughput might only increase sublinearly in the number of AMM contracts. See more details in the full version [22].

5.3 SAMM Equilibrium

We analyze the SPNE of the above game. We only provide a roadmap here, from trader strategy to liquidity provider strategy; the complete analysis is in the full version [22].

5.3.1 Trader Strategy

Consider the case that the system state is $\mathbf{R} = ((R_1^A, R_1^B, R_1^S), \dots, (R_n^A, R_n^B, R_n^S))$. As discussed in Section 4.4, the SAMM gross amount satisfies the c -non-splitting property and c -smaller-better property for a certain $0 < c < 1$. We assume that the required amount of *token A*, b^{BA} , is at most a fraction c of the amount of deposited *token A* in all shards, i.e., $\forall 1 \leq i \leq n, b^{BA} \leq cR_i^A$.

The c -non-splitting property and c -smaller-better property give a trader the incentive to randomly select one of the smallest shards to trade all her required tokens. Recall that $a_i^{BA}(b^{BA})$ is the action of acquiring all b^{BA} *token A* in *shard_i* (Equation 11). We define the set of actions that trade in one of the smallest shards:

► **Definition 5.3.** *The Smallest Shard Action Set is the set of actions that acquire all b^{BA} *token A* in one of the smallest shards under state $\mathbf{R}$: $\mathcal{A}_{1,\min}(b^{BA}, \mathbf{R}) = \{a_i^{BA}(b^{BA}) | \forall j, R_i^A \leq R_j^A\}$.*

The cardinality of $\mathcal{A}_{1,\min}(b^{BA}, \mathbf{R})$ is the number of smallest shards in $\mathbf{R}$. We denote this by $n_{\min}(\mathbf{R}) = |\mathcal{A}_{1,\min}(b^{BA}, \mathbf{R})|$.

We prove that randomly selecting a shard to trade without splitting is a dominant strategy for the trader, and it is strictly better than splitting or trading in a larger shard:

► **Theorem 5.4.** *In $\Gamma_n(tf_{SAMM})$, considering the following dominant strategy of the BA trader which randomly selects one of the smallest shards to acquire all required tokens:*

$$\tau^{BA}(\mathbf{R}, b^{BA}, a^{BA}) = \begin{cases} \frac{1}{n_{\min}(\mathbf{R})}, & \text{if } a^{BA} \in \mathcal{A}_{1,\min}(b^{BA}, \mathbf{R}) \\ 0, & \text{otherwise} \end{cases},$$

then for all strategies π^{BA} that have a positive probability of actions not trading in one of the smallest shards, i.e., $\exists a^{BA} = (b_1^{BA}, \dots, b_i^{BA}, \dots, b_n^{BA}) \notin \mathcal{A}_{1,\min}(b^{BA}, \mathbf{R})$, $\pi^{BA}(\mathbf{R}, b^{BA}, a^{BA}) > 0$, the utility of the BA trader is strictly lower than with strategy τ^{BA} :

$$U^{BA}(\tau^{BA}, \mathbf{R}, b^{BA}) > U^{BA}(\pi^{BA}, \mathbf{R}, b^{BA}).$$

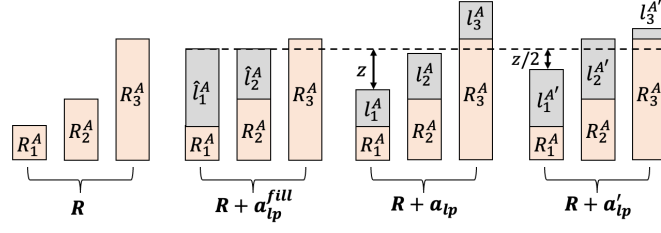
Proof Sketch. Due to the c -non-splitting property, trading in a single shard is strictly better than trading in multiple shards. Then the revenue of trading in one of the smallest shards is strictly better than that in any other shard due to the c -smaller-better property. The complete proof is in the full version [22]. ◀

If shards are balanced, which means all shards have the same amount of deposited tokens, we have $n_{\min}(\mathbf{R}) = n$. Then it is a dominant strategy for the trader to randomly select one of the n shards to trade, as we intended:

► **Corollary 5.5.** *In $\Gamma_n(tf_{SAMM})$, the system state is $\mathbf{R} = ((R_1^A, R_1^B, R_1^S), \dots, (R_n^A, R_n^B, R_n^S))$. If $\forall i, j, R_i^A = R_j^A$ and $R_i^B = R_j^B$, then the perfect parallelism strategy*

$$\hat{\tau}^{BA}(\mathbf{R}, b^{BA}, a^{BA}) = \begin{cases} \frac{1}{n}, & \text{if } a^{BA} \in \mathcal{A}_1(b^{BA}) \\ 0, & \text{otherwise} \end{cases}$$

is a dominant strategy for the BA trader.



■ **Figure 2** An example construction of a_{lp}' .

5.3.2 Liquidity Provider Strategy

Having shown that traders behave as intended if the shards are balanced, we consider the strategies of liquidity providers in equilibrium. They should maintain the shard balance. Moreover, their incentives should rebalance the system state even in the face of attacks that break this balance.

We want a liquidity provider to fill up smaller shards to keep shards balanced. We call such an action the *fillup action*, where if the liquidity provider adds tokens to a shard, then the shard is the smallest shard after this action. We denote the fillup action by $a_{lp}^{fill}(\mathbf{R}, l^A, l^B)$. Note that there is only one such action.

► **Definition 5.6.** The fillup action of a liquidity provider $a_{lp}^{fill}(\mathbf{R}, l^A, l^B) = \left((\hat{l}_1^A, \hat{l}_1^B), \dots, (\hat{l}_n^A, \hat{l}_n^B) \right)$ is the action where if the liquidity provider adds tokens to a shard, then the shard is one of the smallest shards after this action:

$$\forall 1 \leq i \leq n : (1) \hat{l}_i^A \geq 0, \sum_{i=1}^n \hat{l}_i^A = l^A, (2) \forall \hat{l}_i^A > 0, \forall j : \hat{l}_i^A + R_i^A \leq \hat{l}_j^A + R_j^A.$$

The *fillup strategy* only uses the fillup action:

► **Definition 5.7.** The fillup strategy of a liquidity provider $\tau_{lp}^{fill}(\mathbf{R}, l^A, l^B)$ is the strategy that only takes the fillup action:

$$\tau_{lp}^{fill}(\mathbf{R}, l^A, l^B, a_{lp}) = \begin{cases} 1, & \text{if } a_{lp} = a_{lp}^{fill}(\mathbf{R}, l^A, l^B) \\ 0, & \text{otherwise} \end{cases}.$$

We first prove that, when all shards have identical sizes, the fillup action is to add tokens to all shards evenly, which is the best response of the liquidity provider:

► **Theorem 5.8.** Denote by $\hat{a}_{lp} = \left(\left(\frac{1}{n} l^A, \frac{1}{n} l^B \right), \dots \right)$ the action of evenly depositing tokens in all shards. In $\Gamma_n(tf_{SAMM})$, if for all i and j that the liquidity amounts are the same, $R_i^A = R_j^A$ and $R_i^B = R_j^B$, the liquidity provider strategy which only takes action $\hat{a}_{lp}$,

$$\tau_{lp}(\mathbf{R}, l^A, l^B, a_{lp}) = \begin{cases} 1, & \text{if } a_{lp} = \hat{a}_{lp} \\ 0, & \text{otherwise} \end{cases}$$

and any best response of the trader constitutes an SPNE.

Proof Sketch. Traders prefer trading in smaller shards to reduce their costs. At the same time, fees are higher in larger shards. So the liquidity provider should increase her share in the smallest shards. This dual objective is optimally achieved by uniformly distributing tokens across all shards. The complete proof is in the full version [22]. ◀

The above theorem indicates that once the system reaches a balanced state, this state is stable. We now show that even if the system reaches an unbalanced state, maybe due to an attacker, it converges to a balanced state since the liquidity provider uses the fillup strategy.

We show that the fillup strategy is the only best response in all SPNE:

► **Theorem 5.9.** *In $\Gamma_n(tf_{SAMM})$, in all SPNE, the liquidity provider’s best response is the fillup strategy:*

$$\tau_{lp}^{fill}(\mathbf{R}, l^A, l^B, a_{lp}) = \begin{cases} 1, & \text{if } a_{lp} = a_{lp}^{fill}(\mathbf{R}, l^A, l^B) \\ 0, & \text{otherwise} \end{cases}.$$

Proof Sketch. Given any action a_{lp} that is not the fillup action, we can construct a new action a'_{lp} that is strictly better than a_{lp} . By ensuring the smallest shard in $\mathbf{R} + a'_{lp}$ is larger than that in $\mathbf{R} + a_{lp}$, we increase trading fees garnered from each transaction. Moreover, this smallest shard is also the smallest before the action, maximizing the liquidity provider’s share. Consequently, the liquidity provider earns higher revenue under a'_{lp} than under a_{lp} . Thus, any strategy incorporating an action other than the fillup action is not optimal. Figure 2 illustrates an example of the a'_{lp} construction. The complete proof is in the full version [22]. ◀

Theorem 5.9 does not prove the existence of an SPNE. We prove one exists by constructing an SPNE for a general (perhaps unbalanced) starting state (see the full version [22]): Liquidity providers follow the fillup strategy, and traders use the smallest shard with the largest share of the previous liquidity provider.

In summary, we showed that the system achieves a stable state with perfect parallelism. After events that create heterogeneous shard sizes (like imbalancing liquidity additions by an attacker or asymmetric liquidity removals by liquidity providers), liquidity providers rebalance shards once sufficient liquidity has been provided. Malicious attackers incur ongoing losses, and when their budget is exhausted, the system returns to a balanced state. These results indicate that SAMM is robust to such attacks.

6 Evaluation

To evaluate the performance, we use the state-of-the-art blockchains Sui (§6.1) and Solana. We find that the throughput of a single-contract AMM is limited (§6.2), and that it improves with the number of SAMM shards (§6.3). Our analysis shows that further improvement is possible by increasing the platform’s parallelism (§6.4).

6.1 Experimental Setup

We conduct experiments on the Sui [14] blockchain. Smart contracts in Sui are independent *objects*, and Sui executes transactions on different objects in parallel. We implement SAMM⁴ in the Move language [13]. We deploy a local testnet, which follows the default configuration, consisting of 4 validators maintaining consensus of the blockchain. The latency of transactions is always higher than 1 second due to Sui’s consensus protocol. We issue transactions using Sui’s Rust RPC interface.

⁴ <https://github.com/MountainGold/SAMM-Sui-Evaluation>

As a baseline, we test the latency of simple token transfers. As expected, unencumbered by smart-contract coordination constraints, the latency is consistently smaller than 200 msec (Figure 1) in Sui at 2360tps (outside the figure range). This throughput is about twice the maximum rate observed in our AMM experiments on Sui. The latency remains under one second because token transfers in Sui do not require immediate consensus among validators.

We also conduct experiments on Solana [88], which has a different architecture and consensus protocol. Details of the Solana experimental setup are in the full version [22].

For all reported results, we use a machine with 2 TB of memory and 256 CPU cores. We run 100 trader processes for Sui (and 50 for Solana). Several experiments with more traders or with the testnet deployed and transactions sent from separate machines produce similar results. This shows that workload generation does not introduce bottlenecks.

Each trader sends transactions at random intervals following an exponential distribution with an expected frequency of λ . A trader can send another transaction before the previous one is confirmed. Note this experiment is only for performance evaluation, so traders follow the perfect parallelism strategy. We vary the overall frequency of transactions by setting different values of the individual λ values. In each test, we set a target throughput. We first warm up the system by sending transactions for 500 seconds and then measure actual throughputs and latencies for the following 100 seconds. If the latency is stable during the 100 measurement seconds, we report the mean value.

6.2 Single-Contract Bottleneck

To demonstrate the bottleneck of a single AMM, we first deploy a standard CPMM. We use the OmniSwap contract [64] in Sui and the token-swap contract from the Solana Program Library [53], which are generalizations of Uniswap v2.

For the Sui experiment, Figure 1 ($n = 1$) shows latencies of transaction processing (Y axis) in workloads with varying transaction frequencies (X axis) using a single OmniSwap contract. We test each frequency 5 times and calculate the truncated average, excluding the two extreme values. Error bars show all measured values. The average latency increases gradually with the transaction frequency up to 214tps before crossing the 3-second line. With higher frequencies, transactions frequently fail.

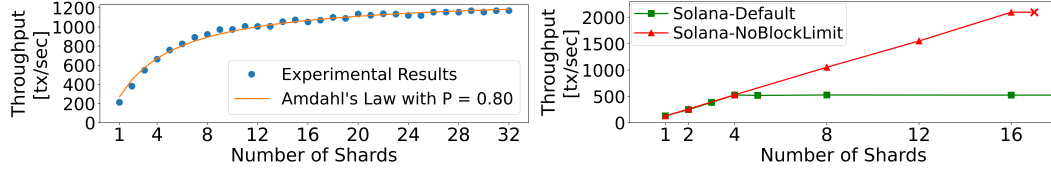
Our Solana experiments produce similar results (see the full version [22]). The throughput bottleneck of a single CPMM contract in both Solana-Default and Solana-NoBlockLimit is 129tps.

6.3 SAMM Evaluation

We implement SAMM by modifying the fee mechanism of CPMM contracts and deploying a varying number of shards (contracts). Each trader randomly selects a shard to trade.

Figure 1 reports the average latency under varying demand (tps) across different shard counts in our Sui experiments. The latency in the case of one SAMM contract is indistinguishable from a single CPMM contract. In some instances, more than half of the transactions failed, which is marked with $\times$ in the graph. As the number of SAMM contracts increases, the system can process higher demand.

To quantify SAMM's performance enhancements in Sui, we evaluate the throughput with a varying number of shards. We set a latency cap of 3 seconds. As depicted in Figure 1, the latency rises rapidly once it surpasses 2 seconds. Therefore, choosing other latency caps beyond 2 seconds does not significantly affect the results. The throughput with n shards is thus the highest frequency that produced a latency lower than 3 seconds. Figure 3a shows the



(a) SAMM's experimental throughput in Sui.

(b) SAMM's experimental throughput in Solana.

■ **Figure 3** Maximal throughput as a function of the number of SAMM shards in Sui and Solana.

throughput increases almost linearly in the beginning and then converges to a bound. With 32 shards, the maximal throughput exceeds $1185tps$, more than five times the throughput of a single OmniSwap contract.

On Solana, we test two configurations: *Solana-Default* uses the standard mainnet gas limits, while *Solana-NoBlockLimit* removes the per-block gas cap to expose the architecture's limit (see the full version [22]). Figure 3b shows that the maximal throughput scales almost linearly up to 4 shards under *Solana-Default* and up to 16 shards under *Solana-NoBlockLimit*, with 17 shards failing to match the tps of 16 shards, marked with $\times$.

6.4 Parallelization in the Underlying Platform

When there are n concurrently operating AMM shards, each with a maximum throughput of $T_{\max}$, the total maximum throughput, $T_{\text{total}}(n)$, is influenced by the fraction P of the transaction that can be parallelized, according to Amdahl's Law. This law states that the speedup ratio, $S(n)$, is the total throughput relative to a single AMM's throughput, according to the expression $S(n) = \frac{1}{(1-P) + \frac{P}{n}}$. For fully sequential systems like Ethereum, $P = 0$, resulting in no throughput gain. For stability, Solana sets a gas limit that keeps throughput well below its serial bottleneck. Its throughput is thus linear in n , up to the artificial limit (Figure 3b).

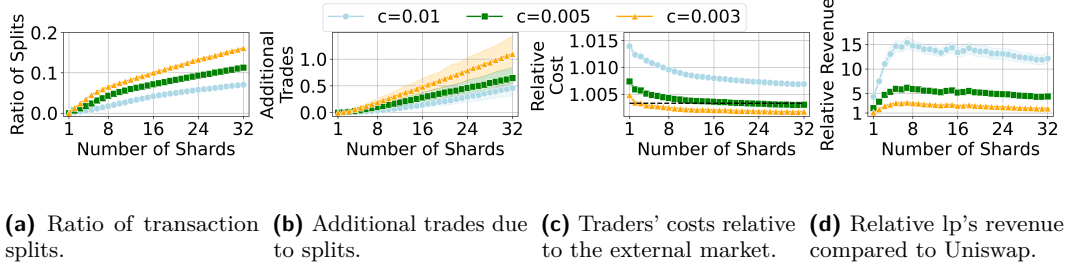
In Sui, we fit the experimental data with Amdahl's law and conclude the parallelizable part of a transaction is $P = 0.80$ ($R^2 = 0.99$). Since the serial components of transactions are invariant, throughput improvements are inherently limited. According to the fitted curve, the throughput is bounded by $1330tps$. We find that if transactions in Sui have a larger parallelizable portion, the throughput improvement due to SAMM is even better (see the full version [22]).

7 Incentive-Compatibility Validation and Secondary Effects

We simulate SAMM on real trading data (§7.1): Incentive compatibility holds even without assuming perfect arbitrageurs (§7.2). SAMM also raises liquidity-provider revenue at trader costs comparable to a standard CPMM, without incentivizing sandwich attacks or amplifying losses and overhead under price fluctuations (§7.3).

7.1 Simulation Setup

We simulate SAMM with workloads from public CPMM trading data, namely the Uniswap v2 Ethereum AMM for the pair of tokens USDC and ETH (hereinafter the *workload*). We assume both Uniswap and SAMM operate at the maximal throughput measured in our performance evaluation. In each test, we simulate over 1 second, following a 1-second warm-up period. We repeat each simulation 100 times and calculate averages. Note that a 1-second



■ **Figure 4** Trade splits, trader costs, and lp's (liquidity providers') revenue in SAMM.

simulation corresponds to several hours' trading in the real world. We conservatively choose the throughput of SAMM according to the performance evaluation of Sui where throughput improves sublinearly with the number of shards.

We simulate SAMM using 1 to 32 shards and Uniswap. For SAMM with n shards, we evenly distribute the liquidity of the workload among the n shards to establish the initial state of SAMM. We test three different values of c , namely 0.003, 0.005, and 0.01. We still do not consider the gas fee in the simulation, as it would only make the results better since it discourages trade splitting (§5.1).

In each test, we replay the trades in the workload. Each trader selects the shard offering the lowest price and may split the trade into several smaller trades if this reduces her costs.

Our simulation differs from the game-theoretic analysis in two key aspects. First, the simulation does not assume perfect arbitrageurs, as there are no synthetic arbitrage transactions. Second, the trade amount of some transactions exceeds the c fraction of a single shard's liquidity. This is particularly relevant in our simulation, where each shard holds only $1/n$ of the total liquidity, occasionally resulting in trade splits. See more details in the full version [22].

7.2 Incentive-Compatibility Validation

We analyze the proportion of trades that are split into multiple transactions within SAMM. With $c = 0.01$, trade splits are infrequent, occurring in less than 8% of trades, even with 32 shards (Figure 4a). Furthermore, the total number of trades (the trader might split a trade into more than 2 parts) results in less than a 45% increase in overall trade count (Figure 4b), a relatively minor increment given the five-fold throughput improvement. When the number of shards is larger or c is smaller, the proportion of trade splits is higher. We also confirm that the distribution of trades in SAMM is balanced (see the full version [22]). This confirms that our theoretical analysis holds without assumptions of perfect arbitrageurs or trade size limits.

7.3 Secondary Effects of Sharding

Sharding results in smaller shards, since liquidity is split. This results in higher slippage for trades. We analyze how this affects participant revenues and AMM vulnerabilities.

Traders' Costs and Liquidity Provider Revenue. We now analyze how sharding affects the revenues of traders and liquidity providers. In a trade with a set output amount and fixed deposited amounts of tokens in a shard, adjusting the fee parameters is zero-sum for the trader and liquidity provider: with higher fees, the liquidity provider gains more revenue,

while the trader incurs higher costs (Equation 2). However, sharding allows more trades to be processed, so liquidity providers can gain more revenue while each trader’s cost still remains comparable to that in a standard CPMM.

Our simulation also validates this economic impact. Figure 4c illustrates traders’ costs: while increasing with the fee parameter c , costs decrease as the number of shards increases, remaining comparable to Uniswap or even lower. This shows how higher throughput can offset increased slippage. Figure 4d reveals a significant increase in total liquidity provider revenue; it climbs with the number of shards (reaching up to 15 times that of Uniswap) due to more trades, before eventually declining as individual fees per trade diminish in very small shards. These trends demonstrate a key advantage of SAMM: It increases the exchange volume, benefiting all participants, and is not a zero-sum game that simply reallocates costs. Hence, SAMM attracts participants when competing with other AMMs. More mathematical analysis and empirical details are provided in the full version [22].

We also observe that SAMM outperforms Uniswap v2 on *volume capacity*, which measures the ability of handling large trades with small price impact (see the full version [22]).

The analysis of revenue, volume capacity and incentive-compatibility (§7.2) implies a trade-off between participant incentives and system performance. A larger c results in fewer trade splits and higher revenue for liquidity providers, but the cost for traders increases. Fixing c and increasing the number of shards results in lower trader costs but more trade splits, which can negatively impact system performance. Additionally, liquidity provider revenue peaks at a certain number of shards. System designers can tune c and the number of shards to balance participant incentives and system performance.

Sandwich Attacks and Price-Fluctuation Losses and Overhead. A critical consideration is whether the increased slippage inherent in smaller shards exacerbates known AMM vulnerabilities, specifically sandwich attacks and losses due to price fluctuations. Our analysis demonstrates that, counter-intuitively, the profitability of sandwich attacks decreases as liquidity decreases, rendering smaller shards less sensitive (see the full version [22]). When the price of tokens in the external market fluctuates, arbitrageurs can arbitrage between the external market and the AMM, causing losses for the liquidity providers, called loss-versus-rebalancing (LVR) [59, 36, 58]. For CPMM-based AMMs like SAMM, with perfect arbitrageurs, the loss per unit of liquidity is independent of the size of the AMM [59, Example 3], meaning the loss ratio is unaffected by sharding. Thus, SAMM enhances throughput without amplifying these economic risks.

Note that the fluctuation of token prices in the external market may lead to arbitrage opportunities, and with more shards there are more such opportunities. However, in the worst case, each external price move triggers one arbitrage per shard per block. An n -shard system (say, $n = 32$) delivering hundreds of *tps* would suffer at most n arbitrage transactions per block. This overhead is minor.

8 Conclusion

We present SAMM, a scalable AMM that employs smart contract sharding. SAMM addresses incentive-driven security challenges through a novel trading fee function, which we validate via game-theoretic analysis and simulations on real trade traces. We implement and deploy SAMM on local Sui and Solana testnets, demonstrating throughput improvements of over 5x and 16x, respectively, up to the limits of the underlying platforms. SAMM is directly applicable and has already attracted industry interest. Nonetheless, how to choose and adapt

the fee parameters and the threshold c as trade-size distributions shift, and how sensitive the system is to their miscalibration, remain to be studied and verified in real-world deployments. Our results provide the first demonstration that high-demand smart contracts can be securely scaled via sharding. They also highlight the importance of minimizing serial bottlenecks in smart-contract platforms to enable further scaling (see the full version [22]).

References

- 1 Ittai Abraham, Danny Dolev, Ittay Eyal, and Joseph Y Halpern. Colordag: An incentive-compatible blockchain. In *37th International Symposium on Distributed Computing (DISC 2023)*, pages 1:1–1:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.DISC.2023.1.
- 2 Austin Adams. Layer 2 be or layer not 2 be: Scaling on uniswap v3. <https://arxiv.org/abs/2403.09494>, 2024.
- 3 Hayden Adams. Introducing the uniswapx protocol. <https://blog.uniswap.org/uniswapx-protocol>, 2023.
- 4 Hayden Adams, Mark Toda, Alex Karys, Xin Wan, Daniel Gretzke, Eric Zhong, Zach Wong, Daniel Marzec, Robert Miller, Hasu, Karl Floersch, and Dan Robinson. Unichain. https://developers.uniswap.org/whitepaper_unichain.pdf, 2024.
- 5 Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 core. <https://app.uniswap.org/whitepaper.pdf>, 2020.
- 6 Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. <https://app.uniswap.org/whitepaper-v3.pdf>, 2021.
- 7 Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 80–91, 2020. doi:10.1145/3419614.3423251.
- 8 Guillermo Angeris, Alex Evans, Tarun Chitra, and Stephen Boyd. Optimal routing for constant function market makers. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, pages 115–128, 2022. doi:10.1145/3490486.3538336.
- 9 Mohammad Ali Asef and Seyed Mojtaba Hosseini Bamakan. From $x^* y = k$ to uniswap hooks: A comparative review of decentralized exchanges (dex). <https://arxiv.org/abs/2410.10162>, 2024.
- 10 Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019. doi:10.1145/3319535.3363213.
- 11 Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. Maximizing extractable value from automated market makers. In *International Conference on Financial Cryptography and Data Security*, pages 3–19. Springer, 2022. doi:10.1007/978-3-031-18283-9_1.
- 12 Mihailo Bjelic, Sandeep Nailwal, Amit Chaudhary, and Wenxuan Deng. Pol: One token for all polygon chains. <https://polygon.technology/papers/pol-whitepaper>. Accessed, April 2024.
- 13 Sam Blackshear, Evan Cheng, David L Dill, Victor Gao, Ben Maurer, Todd Nowacki, Alistair Pott, Shaz Qadeer, Dario Russi Rain, Stephane Sezer, et al. Move: A language with programmable resources. <https://developers.diem.com/papers/diem-move-a-language-with-programmable-resources/2019-06-18.pdf>.
- 14 Sam Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, et al. Sui lutris: A blockchain combining broadcast and consensus. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2606–2620, 2024. doi:10.1145/3658644.3670286.

- 15 Lee Bousfield, Rachel Bousfield, Chris Buckland, Ben Burgess, Joshua Colvin, Edward W. Felten, Steven Goldfeder, Daniel Goldman, Braden Huddleston, Harry Kalodner, Frederico Arnaud Lacs, Harry Ng, Aman Sanghi, Tristan Wilson, Valeria Yermakova, and Tsahi Zidenberg. Arbitrum nitro: A second-generation optimistic rollup. <https://github.com/OffchainLabs/nitro/blob/master/docs/Nitro-whitepaper.pdf>. Accessed, April 2024.
- 16 Lea Salome Brugger, Laura Kovács, Anja Petkovic Komel, Sophie Rain, and Michael Rawson. Checkmate: Automated game-theoretic security reasoning. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1407–1421, 2023. doi:10.1145/3576915.3623183.
- 17 Bucket Protocol. Bucket Protocol Documentation. <https://docs.bucketprotocol.io/>. Accessed, December 28, 2025.
- 18 Andrea Canidio and Robin Fritsch. Batching trades on automated market makers. In *5th Conference on Advances in Financial Technologies (AFT 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.AFT.2023.24.
- 19 Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 154–167, 2016. doi:10.1145/2976749.2978408.
- 20 TH Chan, Ke Wu, and Elaine Shi. Mechanism design for automated market makers. In *7th Conference on Advances in Financial Technologies (AFT 2025)*, pages 7:1–7:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.AFT.2025.7.
- 21 Hongyin Chen, Yubin Ke, Xiaotie Deng, and Ittay Eyal. Prrr: Personal random rewards for blockchain reporting. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 3080–3098. IEEE, 2026. doi:10.1109/SP63933.2026.00136.
- 22 Hongyin Chen, Amit Vaisman, and Ittay Eyal. Samm: Sharded automated market maker. <https://arxiv.org/abs/2406.05568>, 2024.
- 23 Hongyin Chen, Xu Zheng, Jichen Li, and Ittay Eyal. Time is money: Incentivized causal transaction ordering. arXiv preprint, 2026. doi:10.48550/arXiv.2607.11496.
- 24 Tarun Chitra, Guillermo Angeris, and Alex Evans. Differential privacy in constant function market makers. In *International Conference on Financial Cryptography and Data Security*, pages 149–178. Springer, 2022. doi:10.1007/978-3-031-18283-9_8.
- 25 Christoph Rosenmayr and Mateusz Dominiak. Statistical arbitrage on AMMs and block building on Ethereum – Part 1. <https://greenfieldcapital.com/2024/09/10/statistical-arbitrage-on-amms-and-block-building-on-ethereum-part-1/>, 2024.
- 26 Curve.fi. Curve documentation release 1.0.0. https://curve.readthedocs.io/_/downloads/en/latest/pdf/, 2022.
- 27 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020. doi:10.1109/SP40000.2020.00040.
- 28 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022. doi:10.1145/3492321.3519594.
- 29 DefiLlama. Total value locked all chains. <https://defillama.com/chains>. Accessed, July 2024.
- 30 Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. Adding concurrency to smart contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 303–312, 2017. doi:10.1145/3087801.3087835.
- 31 Ellipsis Labs. phoenix-cli. <https://github.com/Ellipsis-Labs/phoenix-cli>. Accessed, December 28, 2025.

- 32 Daniel Engel and Maurice Herlihy. Composing networks of automated market makers. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 15–28, 2021. doi:10.1145/3479722.3480987.
- 33 Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, pages 45–59, 2016. URL: <https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal>.
- 34 Matheus VX Ferreira, Daniel J Moroz, David C Parkes, and Mitchell Stern. Dynamic posted-price mechanisms for the blockchain transaction-fee market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 86–99, 2021. doi:10.1145/3479722.3480991.
- 35 Emilio Frangella and Lasse Herskind. Aave V3 Technical Paper. https://raw.githubusercontent.com/aave/aave-v3-core/master/techpaper/Aave_V3_Technical_Paper.pdf, 2022.
- 36 Robin Fritsch and Andrea Canidio. Measuring arbitrage losses and profitability of amm liquidity. In *Companion Proceedings of the ACM on Web Conference 2024*, pages 1761–1767, 2024. doi:10.1145/3589335.3651961.
- 37 Péter Garamvölgyi, Yuxi Liu, Dong Zhou, Fan Long, and Ming Wu. Utilizing parallelism in smart contracts on decentralized blockchains by taming application-inherent conflicts. In *Proceedings of the 44th International Conference on Software Engineering*, pages 2315–2326, 2022. doi:10.1145/3510003.3510086.
- 38 Bikramaditya Ghosh, Hayfa Kazouz, and Zaghun Umar. Do automated market makers in defi ecosystem exhibit time-varying connectedness during stressed events? *Journal of Risk and Financial Management*, 16(5):259, 2023. doi:10.3390/jrfm16050259.
- 39 Mohak Goyal, Geoffrey Ramseier, Ashish Goel, and David Mazières. Finding the right curve: Optimal design of constant function market makers. In *Proceedings of the 24th ACM Conference on Economics and Computation*, pages 783–812, 2023. doi:10.1145/3580507.3597688.
- 40 Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 201–226. Springer, 2020. doi:10.1007/978-3-030-51280-4_12.
- 41 Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019. doi:10.1109/DSN.2019.00063.
- 42 Lioba Heimbach and Roger Wattenhofer. Eliminating sandwich attacks with the help of game theory. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 153–167, 2022. doi:10.1145/3488932.3517390.
- 43 Charlie Hou, Mingxun Zhou, Yan Ji, Phil Daian, Florian Tramèr, Giulia Fanti, and Ari Juels. Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning. In *28th Annual Network and Distributed System Security Symposium (NDSS)*, 2021. doi:10.14722/ndss.2021.24188.
- 44 Hubble Protocol. Hubble Protocol Documentation. <https://docs.hubbleprotocol.io/>. Accessed, December 28, 2025.
- 45 Kamino Finance. Kamino Lend Litepaper: Unified Liquidity Market. <https://kamino.com/docs>. Accessed, July 25, 2026.
- 46 Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE symposium on security and privacy (SP)*, pages 583–598. IEEE, 2018. doi:10.1109/SP.2018.000-5.

- 47 Kshitij Kulkarni, Theo Diamandis, and Tarun Chitra. Routing mev in constant function market makers. In *International Conference on Web and Internet Economics*, pages 456–473. Springer, 2023. doi:10.1007/978-3-031-48974-7_26.
- 48 L2 Lab. Zkswap: A layer-2 token swap protocol based on zk-rollup. https://github.com/l2labs/zkswap-whitepaper/blob/master/zkswap_en.pdf, 2020.
- 49 Aptos Labs. The aptos blockchain: Safe, scalable, and upgradeable web3 infrastructure. https://aptosfoundation.org/whitepaper/aptos-whitepaper_en.pdf, 2022.
- 50 Alfred Lehar, Christine Parlour, and Marius Zoican. Fragmentation and optimal liquidity supply on decentralized exchanges. <https://arxiv.org/abs/2307.13772>, 2023.
- 51 Chenxin Li, Peilun Li, Dong Zhou, Zhe Yang, Ming Wu, Guang Yang, Wei Xu, Fan Long, and Andrew Chi-Chih Yao. A decentralized blockchain with high throughput and fast confirmation. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pages 515–528, 2020. URL: <https://www.usenix.org/conference/atc20/presentation/li-chenxing>.
- 52 Zihao Li, Jianfeng Li, Zheyuan He, Xiapu Luo, Ting Wang, Xiaoze Ni, Wenwu Yang, Xi Chen, and Ting Chen. Demystifying defi mev activities in flashbots bundle. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 165–179, 2023. doi:10.1145/3576915.3616590.
- 53 Solana Program Library. Token swap program. <https://spl.solana.com/>. Accessed, July 2024.
- 54 Jian Liu, Peilun Li, Raymond Cheng, N Asokan, and Dawn Song. Parallel and asynchronous smart contract execution. *IEEE Transactions on Parallel and Distributed Systems*, 33(5):1097–1108, 2021. doi:10.1109/TPDS.2021.3095234.
- 55 MK Manoylov. Arbitrum surpasses \$150 billion in total transaction volume on uniswap. <https://www.theblock.co/post/292655/>, 2024.
- 56 Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers 22*, pages 3–18. Springer, 2019. doi:10.1007/978-3-662-58820-8_1.
- 57 Nicolas Michel, Mohamed E Najd, and Ghada Almashaqbeh. ammbost: State growth control for amms. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 143–156. IEEE, 2025. doi:10.1109/DSN64029.2025.00027.
- 58 Jason Milionis, Ciamac C Moallemi, and Tim Roughgarden. Automated market making and arbitrage profits in the presence of fees. In *International Conference on Financial Cryptography and Data Security*, pages 159–171. Springer, 2024. doi:10.1007/978-3-031-78676-1_9.
- 59 Jason Milionis, Ciamac C Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing. <https://arxiv.org/abs/2208.06046>, 2022.
- 60 Michael Mirkin, Yan Ji, Jonathan Pang, Arian Klages-Mundt, Ittay Eyal, and Ari Juels. Bdos: Blockchain denial-of-service. In *Proceedings of the 2020 ACM SIGSAC conference on Computer and Communications Security*, pages 601–619, 2020. doi:10.1145/3372297.3417247.
- 61 Morpho. Morpho documentation: Market. <https://docs.morpho.org/learn/concepts/market/>. Accessed, December 24, 2025.
- 62 Marwa Mouallem, Lorenz Breidenbach, Ittay Eyal, and Ari Juels. Resilient alerting protocols for blockchains. To appear in the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2026. doi:10.48550/arXiv.2602.10892.
- 63 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>, 2008.
- 64 OmniBTC. Omniswap. <https://github.com/OmniBTC/OmniSwap>. Accessed, April 2024.
- 65 George Pirlea, Amrit Kumar, and Ilya Sergey. Practical smart contract sharding with ownership and commutativity analysis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 1327–1341, 2021. doi:10.1145/3453483.3454112.

- 66 QuickSwap. Quicksnap documentation. <https://docs.quicksnap.exchange/>. Accessed: April 2024.
- 67 risechain. pevsn. <https://github.com/risechain/pevsn>. Accessed, December 23, 2025.
- 68 Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. <https://arxiv.org/abs/1906.08936>, 2019.
- 69 Tim Roughgarden. Transaction fee mechanism design. *ACM SIGecom Exchanges*, 19(1):52–55, 2021. doi:10.1145/3476436.3476445.
- 70 Jan Christoph Schlegel, Mateusz Kwasnicki, and Akaki Mamageishvili. Axioms for constant function market makers. In *Proceedings of the 24th ACM Conference on Economics and Computation*, page 1079, 2023. doi:10.1145/3580507.3597720.
- 71 Reinhard Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfrageträgheit: Teil i: Bestimmung des dynamischen preisgleichgewichts. *Zeitschrift für die gesamte Staatswissenschaft/Journal of Institutional and Theoretical Economics*, 121(2):301–324, 1965. URL: <https://www.jstor.org/stable/40748884>.
- 72 Ilya Sergey and Aquinas Hobor. A concurrent perspective on smart contracts. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*, pages 478–493. Springer, 2017. doi:10.1007/978-3-319-70278-0_30.
- 73 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022. doi:10.1145/3548606.3559361.
- 74 Sui Foundation. DeepBookV3. <https://docs.sui.io/onchain-finance/deepbook/deepbookv3/deepbook>. Accessed, July 25, 2026.
- 75 suiscan. Average gas fee. <https://suiscan.xyz/mainnet/analytics/avg-gas-fee>. Accessed, September 2024.
- 76 Sushi. Be a crypto chef with sushi. <https://www.securities.io/sushiswap-whitepaper/>. Accessed: July 2026.
- 77 TheBlock. Sui price (sui): Price index and live chart. <https://www.theblock.co/price/248445/sui-sui-eur>. Accessed, September 2024.
- 78 Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. Mad-htlc: because htlc is crazy-cheap to attack. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1230–1248. IEEE, 2021. doi:10.1109/SP40001.2021.00080.
- 79 Uniswap Labs. Uniswap on Optimism. <https://blog.uniswap.org/uniswap-optimism-alpha>, 2021.
- 80 Heinrich Von Stackelberg. *Market structure and equilibrium*. Springer Science & Business Media, 2010. doi:10.1007/978-3-642-12586-7.
- 81 Jianhuan Wang, Jichen Li, Zecheng Li, Xiaotie Deng, and Bin Xiao. n-mvlt attack: Optimal transaction reordering attack on defi. In *European Symposium on Research in Computer Security*, pages 367–386. Springer, 2023. doi:10.1007/978-3-031-51479-1_19.
- 82 Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX symposium on networked systems design and implementation (NSDI 19)*, pages 95–112, 2019. URL: <https://www.usenix.org/conference/nsdi19/presentation/wang-jiaping>.
- 83 Ye Wang, Yan Chen, Haotian Wu, Liyi Zhou, Shuiguang Deng, and Roger Wattenhofer. Cyclic arbitrage in decentralized exchanges. In *Companion Proceedings of the Web Conference 2022*, pages 12–19, 2022. doi:10.1145/3487553.3524201.
- 84 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. <https://ethereum.github.io/yellowpaper/paper.pdf>, 2015.

- 85 Matheus Venturyne Xavier Ferreira and David C Parkes. Credible decentralized exchange design via verifiable sequencing rules. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 723–736, 2023. doi:10.1145/3564246.3585233.
- 86 Tianjing Xu, Yongqi Zhong, Yiming Zhang, Ruofan Xiong, Jingjing Zhang, Guangtao Xue, and Shengyun Liu. Vegeta: Enabling parallel smart contract execution in leaderless blockchains. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 795–811, 2025. URL: <https://www.usenix.org/conference/nsdi25/presentation/xu-tianjing>.
- 87 Aviv Yaish, Gilad Stern, and Aviv Zohar. Uncle maker:(time) stamping out the competition in ethereum. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–149, 2023. doi:10.1145/3576915.3616674.
- 88 Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13. <https://solana.com/solana-whitepaper.pdf>. Accessed, April 2024.
- 89 Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 931–948, 2018. doi:10.1145/3243734.3243853.
- 90 Ren Zhang and Bart Preneel. Lay down the common metrics: Evaluating proof-of-work consensus protocols’ security. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 175–192. IEEE, 2019. doi:10.1109/SP.2019.00086.
- 91 Ren Zhang, Dingwei Zhang, Quake Wang, Shichen Wu, Jan Xie, and Bart Preneel. NC-max: Breaking the security-performance tradeoff in nakamoto consensus. In *Proceedings 2022 Network and Distributed System Security Symposium*, pages 1–18. NDSS, 2022. doi:10.14722/ndss.2022.24370.
- 92 Yi Zhang, Xiaohong Chen, and Daejun Park. Formal specification of constant product (xy=k) market maker model and implementation. <https://github.com/runtimeverification/publications/blob/main/reports/smart-contracts/Uniswap-V1.pdf>, 2018.
- 93 Liyi Zhou, Kaihua Qin, and Arthur Gervais. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. <https://arxiv.org/abs/2106.07371>, 2021.
- 94 Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 428–445. IEEE, 2021. doi:10.1109/SP40001.2021.00027.