ammBoost: State Growth Control for AMMs

Nicholas Michel, Mohamed E. Najd, and Ghada Almashaqbeh

University of Connecticut
{nicolas.michel, menajd, ghada}@uconn.edu

Abstract. Automated market makers (AMMs) are a form of decentralized cryptocurrency exchanges that have attracted huge interest lately. They are considered a prime example of Decentralized Finance (DeFi) applications, a large category under Web 3.0. Their popularity and high trading activity have resulted in millions of on-chain transactions leading to serious scalability issues in terms of throughput and on-chain state size. Existing scalability solutions, when employed in the context of AMMs, are either ineffective due to their large overhead, or suffer from security and centralization issues.

In this paper, we address these challenges by utilizing a new sidechain architecture as a layer 2 solution, building a system called ammBoost. Our system reduces the amount of on-chain transactions, boosts throughput, and supports blockchain pruning. We devise several techniques to enable layer 2 processing while preserving the correct and secure operation of AMMs. These include a *functionality-split* and *layer 2 traffic summarization* paradigm, an *epoch-based deposit mechanism*, and *pool snapshot-based and delayed token-payout trading*. We also build a proof-of-concept of ammBoost for a Uniswap-inspired use case to empirically evaluate performance. Our experiments show that ammBoost decreases the gas cost by 96.05% and the chain growth by at least 93.42%, and that it can support up to 500x of the daily traffic volume observed for Uniswap in practice.

1 Introduction

Cryptocurrencies and blockchain technology provide an innovative model that led to new applications and research frontiers, as well as reshaping the Internet and its digital services (under what is called Web 3.0). Decentralized Finance (DeFi) is a large category under Web 3.0 in which blockchains are used to transform traditional financial services, which are usually centrally managed, into fully decentralized ones. Many of these systems operate in an open-access model, thus removing market entrance barriers for customers, and enabling a transparent and intermediary-free interaction. Smart contracts strengthen this model by providing an automated way to negotiate contract terms and enforce agreements.

Automated market makers (AMMs) are considered a prime example of DeFi services [71]. They build a platform for automated token trading by establishing liquidity pools for token pairs. An AMM is implemented as a decentralized

application (dApp); a set of smart contracts on top of a smart contract-enabled blockchain—where Ethereum is the dominant choice so far, that support operations for trading and liquidity management, such as swaps, mints, burns, and collects. AMMs are a huge industry with a total monthly trading volume of $46 - $95 billion (during the first half of 2023), and an estimated total market cap of nearly $16 billion as of January 2024 [24]. Many popular AMMs are deployed in practice and widely used. Examples include Uniswap [27], Curve [8], DODO [9], and Sushiswap [22], which during the first half of 2023 commanded around 62-71%, 6.38-14.01%, 4.04-7.16%, and 0.69-3.67% of the top ten AMMs market share, respectively [1].

Challenges. At the same time, AMMs are a huge scalability problem. The popularity and high trading activity of AMMs led to serious efficiency problems since they produce a massive number of on-chain transactions. On the one hand, this increases the underlying blockchain size or storage overhead, and on the second hand, it incurs large (gas) fees. This large workload does not only amplify state storage cost, but also transaction processing/confirmation delays due to the low throughput of blockchains (Ethereum's throughput is around 12 transaction/sec on average [11]).[1]

Concretely, based on the traffic analysis that we conducted for 2023 (see Appendix C), Uniswap V3 users produced 20 million transactions on Ethereum in 2023 (transaction sizes range from 400 bytes to 3000 bytes). This translates to adding around 20.2 GB to the Ethereum blockchain. In its year of deployment (Nov 2018), Uniswap V1 generated 34,000 transactions, and in 2023, Uniswap's various versions on Ethereum generated around 80 million transactions, leading to 231,7% increase in transaction volume. These numbers indicate that the scalability problem of AMMs is amplified over the years. This does not only impact the AMM itself, but also other dApps deployed on the underlying blockchain. Such contention drives users to put in high transaction fees so that miners would prioritize their transactions.

The challenge is to handle this scalability problem *without impacting security* (by, e.g., employing trusted third parties or weakening the security of consensus, which introduces new threat vectors).

Limitations of prior work. Improving blockchain scalability is an active research area. Many solutions have been proposed; some of them target layer 1 improving on the consensus itself, such as sharding [42,53,73], while others target layer 2 allowing for some form of off-chain processing, such as payment channels and networks [43] and rollups [16, 31]. However, when it comes to AMMs, applying these solutions impacts performance and security, and may not even cut the storage cost. In sharding, localized workload division policies are used to reduce cross-shard transactions. For smart contract-enabled blockchains, this means that a dApp (so the whole AMM) will be contained in one shard [33, 66],

[1] A dApp on-chain storage includes the state of the smart contract account of this dApp, *and* the transaction history recorded on the blockchain that produced this state. A misconception about on-chain storage cost is counting only the contract *latest* state while ignoring the permanent on-chain storage of these transactions.

thus parallel processing among shards is not utilized. Others [60] use static analysis to shard dApps by splitting them into commuting functional units that can be executed in any order, and assigning each unit to a shard. This cannot work for AMMs since their (per pool) operation is sequential. Distributing liquidity pools among shards has been proposed in [62], but the reliance on locked cross-shard transactions (to support multi-swaps and arbitraging) may degrade the AMM performance. All these sharding solutions log all transactions on-chain (i.e., the shards), so they do not cut the storage cost.[2]

Layer 2 solutions that allow computations, i.e., beyond just currency transfer as in payment networks [43], have limitations. Optimistic rollups have long contestation periods that may reach one week as in Optimism and Arbitrum [16,49]. Thus, a user cannot act based on the submitted state changes immediately, but has to wait until the end of the contestation period to ensure that the submitted results are valid. Moreover, they have security issues—verifiers (who validate the submitted changes during the contestation period) could be centralized trusted entities [65], while incentive compatibility of non-trusted verifiers is still an open question [5, 17, 56] which may lead to adopting incorrect ledger state changes. Zero-knowledge (ZK)-rollups [21, 38, 39] are costly; proof generation may take several minutes and it becomes worse when attesting to complex transactions. They also may have a long transaction confirmation delay that may reach 24 hours as in zkSync Era [31]. This impacts transaction processing and confirmation delays, forcing the users to wait longer for their transactions to be finalized. Not to mention that many of the used ZK systems require a trusted setup.[3]

Sidechains [36,46,47,51] are another type of layer 2 solutions that can improve scalability. Despite their potential, existing efforts mostly focus on two-way peg, i.e., currency transfer between the sidechain and the mainchain, and all of them are considered independent sidechains. That is, each chain has its own transactions, miners, and tokens. Such independence and the focus on two-way peg limit the performance gains that can be achieved, and do not allow for workload sharing between the chains. Moreover, none of these solutions allow pruning stale records.[4] Other instances in practice resorted to operating the whole AMM on an independent sidechain, i.e., as a separate system from the underlying smart contract-enabled blockchain. For example, Polygon [18] operates a fast EVM-compatible sidechain running along with Ethereum. The whole AMM is run

[2] Ledger pruning for sharding has been proposed in [53, 67] but in the UTXO model, i.e., as in Bitcoin; it is not for the account model (as in Ethereum) that represents smart contract-enabled blockchains on top of which AMMs are deployed.

[3] Another instance of layer 2 solutions resorted to employing a *centralized* settlement party that matches trades and generate ZK proofs to prove settlement correctness [50]. Thus, this solution is a form of ZK-rollups that employs a single settlement party.

[4] In parallel to these academic efforts, many industrial initiatives have explored the utility of sidechains [2, 6, 30, 55, 61, 70]. Most of them focused on independent sidechains and two-way peg. Cosmos [6] allows some form of data exchange via publishing events and commitments on the destination chain. This worsens the storage problem due to data duplicates, and there is no support for blockchain pruning.

on this sidechain and tokens can be transferred to Ethereum using bridges or atomic swap techniques. Isolating an AMM on an independent sidechain impacts composability with other dApps running on the mainchain (which is Ethereum in this case), limits interaction with the mainchain to merely currency transfers, and complicates system design and its security due to the involvement of bridges. Not to mention that this solution just moves the on-chain storage cost to the sidechain, which will have scalability issues on its own as the AMM user population grows.

A new approach. Sidechains seem to have (so far under-utilized) potential in building an effective layer 2 solution to promote scalability. This has been observed in [57], who proposed a framework called chainBoost with a new sidechain architecture that has a mutual-dependence relation with the mainchain, thus permitting workload sharing and arbitrary data exchange, as well as blockchain pruning. chainBoost targets resource markers—Web 3.0 systems that offer decentralized digital services, e.g., Filecoin [12] and Livepeer [15]. chainBoost directs all heavy/frequent service-related traffic to the sidechain, which in turn processes this traffic and produces concise summaries of the state changes that are used to sync the mainchain. Once these summaries are confirmed on the mainchain, the temporary blocks containing the actual transactions on the sidechain are pruned. The empirical results in [57] show substantial performance gains in terms of blockchain size, transaction confirmation delays, and throughput. All of these are achieved without compromising security and while keeping the mainchain as the single truth of the system state.

These advantages motivated us to explore the following: *Can we control the state growth of AMMs, and boost their throughput, in a secure and low-overhead way, and without isolating the AMM on a separate blockchain, using dependent-sidechains?*

1.1 Our Contributions

We answer this question in the affirmative and propose ammBoost; a secure storage control and throughput boosting solution for AMMs. In particular, we make the following contributions.

System design. ammBoost introduces a novel approach for dividing the AMM functionality into two modules: one that resides on the mainchain (i.e., the underlying smart contract-enabled blockchain) and another that is operated by the sidechain. In particular, ammBoost offloads processing most transactions (swaps, mints, collects, and burns) to the sidechain, and minimizes the functionality remaining on the mainchain. The latter is encapsulated in a base smart contract called TokenBank, that manages the actual tokens by tracking only the transaction summaries produced by the sidechain. It also includes all operations that must happen in real-time on the mainchain, such as flash loans.

ammBoost solves several challenges related to applying dependent sidechains to AMMs. The chainBoost framework assumes a mutual-dependency relation between the main and side chains; both operate in the same domain and have the same transaction format, services, and miner population. This is not the case

for ammBoost; the AMM is merely a dApp deployed at the application layer, so it does not modify how the mainchain protocol works. Thus, the mainchain miners do not maintain the sidechain as in chainBoost, even they might not be aware of its existence. However, ammBoost's sidechain is impacted by the mainchain since the tokens and the AMM state (and some core functionalities) are on the mainchain. This means means that ammBoost introduces a unidirectional dependency relation: the sidechain is impacted by the mainchain but not vice versa. Furthermore, the sidechain should process all trading activities without a custody of the actual tokens, and must ensure that only transactions for which issuing users own tokens on the mainchain are accepted.

We resolve these challenges by introducing several techniques. First, we require the AMM to have its *own miners* to maintain the sidechain. So, like any blockchain, these miners are assumed to have a mining power with an honest majority, they need to build a Sybil-resistant identity by, e.g., using a proof-of-stake approach, and they will be rewarded for maintaining the sidechain using, e.g., the AMM native token. Second, we introduce *epoch-based deposits*, where a user has to deposit on the mainchain the anticipated amount of tokens needed to back up her activities (or issued transactions) during an epoch on the sidechain. Third, we introduce *pool snapshot-based and delayed token-payout trading*. That is, the pool token balances are retrieved from the mainchain at the beginning of the epoch, which are used to compute trade prices processed on the sidechain. These balances evolve during the epoch based on the processed transactions processed. Users can use newly accrued tokens in trading since all balances are tracked, but they cannot withdraw the actual tokens directly. This is because the sidechain does not hold actual tokens; token payouts and deposit leftover refunds happen at the end of the epoch when TokenBank is synced.

Security analysis. We analyze the security of ammBoost showing that it preserves the correct and secure operation of the AMM.

Implementation and evaluation. We also build a proof-of-concept implementation for a Uniswap-inspired use case, and conduct experiments to empirically evaluate the performance gains, in terms of blockchain size, confirmation delay, and throughput, that ammBoost can achieve. Our experiments show that ammBoost achieves a 96.05% gas cost reduction and 93.42% chain growth reduction (when compared to a Uniswap version deployed on the Sepolia testnet). Our experiments demonstrate that ammBoost can support large traffic volumes, on the order of up to 500x of Uniswap's daily transaction volume. We also study impact of various configuration parameters on the performance gains of ammBoost.

Although the focus of this work is on controlling the storage overhead and boosting throughput of AMMs, we believe that ammBoost's paradigm can enable more optimizations for AMMs, e.g., integration of privacy-preserving techniques. Also, ammBoost could be beneficial for other DeFi applications, and dApps in general, as it introduces a framework for operating application-specific sidechains interacting with smart contract-enabled blockchains. We leave exploring such directions as part of our future work.

2 Background

We provide an overview of the general functionality of AMMs and the chainBoost framework that we use in the design of ammBoost.

Automated market makers. AMMs build platforms for token trading powered by the users themselves. This is done by establishing liquidity pools such that a pool trades a pair of tokens, say tokens A and B. Users are divided into: clients which could be sellers and buyers, and liquidity providers (LPs). Providing liquidity comes from the sellers themselves since buying token A requires paying the price using token B (and vice versa), and from LPs who deposit tokens in the pool and collect fees in return.

Constant function market makers (CFMM) is a popular implementation choice for computing the trading price in AMMs. This formula keeps the ratio of token reserves, and consequently prices, in the pool as balanced as possible to reduce price slippage. In particular, the price of token A multiplied by the price of token B equals a constant. Let the reserves of tokens A and B, i.e., their total amounts, in the pool be res_A and res_B, respectively, then the price of token A is res_B/res_A and the price of token B is res_A/res_B. Accordingly, for an order trading an amount of token A, amt_A, the amount of token B, amt_B, that this order receives is computed as: $amt_B = res_B - (res_A \cdot res_B)/(res_A + amt_A)$.

At a basic level, an AMM implementation supports several transaction types: for trading, there are (exact input and exact output) swaps, and for liquidity management, there are mints, burns and collects that allow LPs to submit liquidity positions, collect their fees, and withdraw these positions, respectively. AMMs may provide additional services, such as flash loans [69] allowing clients to take advantage of arbitrage opportunities across different platforms. Furthermore, more sophisticated liquidity approaches are being adopted, e.g., concentrated liquidity [44] that enable defining a price range over which liquidity will be applied to address issues related to inefficient use of provided funds.

The functionality of an AMM is commonly implemented as a set of smart contracts on top of a smart contract-enabled blockchain, where Ethereum is the dominant choice so far. These contracts create and manage the liquidity pools, and provide the API needed to interact with the AMM. Residing on a public blockchain led to several financial and security issues [37, 58, 63], e.g., front-running attacks, sandwich attacks, miner/maximal extractable value, etc. Understanding and solving these issues are active research areas. We do not discuss these issues further since they are not the focus of this work; we target the storage cost and throughput of AMMs.

The chainBoost framework. chainBoost [57] is a sidechain-based solution that aims to reduce the blockchain storage footprint and confirmation delays, and boost transaction throughput. It introduces a new sidechain architecture that shares the workload with the mainchain, and enables pruning stale records. As such, this sidechain has a mutual-dependence relation with the mainchain. Transactions are classified into sidechain and mainchain transactions, where all service-related operations that can be summarized go to the sidechain, while the

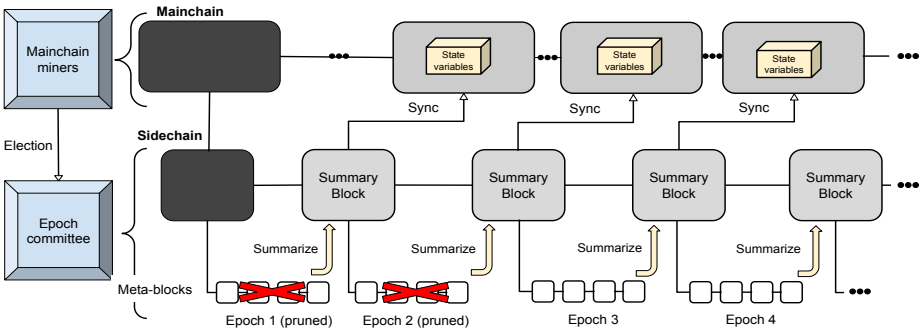

Fig. 1: The chainBoost framework.

rest stay on the mainchain. The sidechain works in parallel to the mainchain, and operates in epochs and rounds (an epoch is ω consecutive rounds and a round is the period during which a new block is mined). At the end of each epoch, the mainchain is synced with summaries of the workload processed by the sidechain in that epoch.

As shown in Figure 1, the sidechain is managed by the mainchain miners, where for each epoch, a committee is elected to process the sidechain traffic during that epoch. The rest of the mainchain miners, who are not on the committee, do not process the sidechain traffic, thus reducing their load. To speedup agreement, chainBoost employs a practical Byzantine fault tolerance (PBFT)-based consensus (similar to those in [48, 52]) for the sidechain.[5]

The sidechain is composed of two types of blocks (as shown in the figure): temporary meta-blocks and permanent summary-blocks. For each sidechain round, the committee mines a meta-block containing the transactions they processed, so that once a transaction appears in a meta-block it is considered final. In the last round of the epoch, this committee mines a summary-block summarizing all state changes induced by the meta-blocks within that epoch. After that, it issues a sync-transaction containing the summarized state changes, which the mainchain miners use to update the relevant state variables on the mainchain. Once the sync-transaction is confirmed on the mainchain, all meta-blocks used to produce the respective summary-block are discarded. This significantly reduces the sidechain size, and subsequently, the mainchain size. At the same time, having permanent summary-blocks allows anyone can verify the source of the state changes recorded on the mainchain.

Applicability of dependent-sidechains to AMMs' setting. Our setting is different from the one in [57]: First, the mainchain and sidechain miners run different protocols. In ammBoost, the mainchain miners belong to a smart contract-enabled blockchain, and the AMM is simply an application deployed on that blockchain. Thus, the sidechain must have its own miner population, to run

[5] Similar to chainBoost, to simplify the presentation, we adopt a leader-based PBFT in which a leader proposes a block for the committee to agree on (as in [52]). Nonetheless, voting-based PBFT (as in [48]) can be used instead.

its protocol, and a technique, such as proof-of-stake, to mitigate Sybil attacks.[6] Second, in chainBoost the two chains are mutually-dependent, i.e., their security and valid operation depend on each other. In ammBoost, the dependence is unidirectional; interruptions on the mainchain impact the sidechain since the base contract that keeps track of the AMM state resides on the mainchain, but not vice versa. Sidechain interruptions will indeed lead to invalid state of the base AMM contract, but this contract resides on the application layer and does not impact the underlying mainchain or other deployed dApps. Third, in ammBoost, the actual token reside on the mainchain, while the trading and liquidity-related activities are handled by the sidechain. Thus, a mechanism is needed to handle token payouts and deposits to enable accepting and processing only valid transactions. Devising techniques to address these issues resemble the core novelty of ammBoost system design.

3 Preliminaries

Notation. We use λ to denote the security parameter, and pp to denote the system public parameters. We use $\mathcal{L}$ to denote a ledger (or blockchain), $\mathcal{L}_{\mathsf{mc}}$ to denote the mainchain ledger, and $\mathcal{L}_{\mathsf{sc}}$ to denote the sidechain ledger. The former is the smart contract-enabled blockchain on top of which the AMM base smart contract is deployed, while the latter is the blockchain of the AMM ecosystem. Each party maintains a secret key sk and a public key pk. Lastly, we use PPT as a shorthand for probabilistic polynomial time.

System model. ammBoost involves a base smart contract representing the AMM on the mainchain, and a sidechain that processes most of the AMM workload. Anyone can join/leave the AMM at anytime, and these parties are known using their public keys. Participants are three types: clients $\mathcal{C}$ who are only interested in using the AMM trading services, liquidity providers $\mathcal{LP}$ who provide liquidity for the pools operated by the AMM, and miners $\mathcal{M}$ who maintain the AMM sidechain. We do not place any restrictions on the mainchain beyond being a secure smart contract-enabled blockchain. ammBoost operates in rounds and epochs (as defined earlier). The sidechain is managed by a committee elected from the sidechain miners, where a new committee is elected for each epoch. This committeee runs a PBFT-based consensus to mine new blocks: temporary meta-blocks that record transactions, and permanent summary-blocks that summarize meta-blocks mined in an epoch. The committee also issues sync-transactions to sync the base AMM smart contract deployed on the mainchain. Accordingly, the ammBoost framework provides the following functionalities:

$\mathsf{SystemSetup}(1^\lambda, \mathcal{L}_{\mathsf{mc}}) \rightarrow (\mathsf{pp}, \mathcal{L}^0_{\mathsf{sc}})$**:** Takes as input the the security parameter λ and the mainchain $\mathcal{L}_{\mathsf{mc}}$. It configures the system public parameters pp, and deploys a base AMM smart contract on $\mathcal{L}_{\mathsf{mc}}$. It outputs pp and the initial

[6] Indeed, a miner can choose to operate on both the mainchain and the sidechain. Still this miner runs two protocols, one for each chain, rather than one protocol.

sidechain ledger state $\mathcal{L}_{\mathsf{sc}}^0$ (which is the genesis block referencing the mainchain block containing the base contract).

$\mathsf{PartySetup}(\mathsf{pp}) \to (\mathsf{state})$**:** Takes as input pp and outputs the initial local state of the party state, which contains a keypair $(\mathsf{sk}, \mathsf{pk})$, and in case of miners, the current view of $\mathcal{L}_{\mathsf{sc}}$.

$\mathsf{CreateTx}(\mathsf{txtype}, \mathsf{aux}) \to (\mathsf{tx})$**:** Takes the transaction type txtype and any additional information aux as inputs, and outputs a transaction tx of one of the following types:

- $\mathsf{tx}_{\mathsf{Deposit}}$: Allows a user to deposit funds on the mainchain to support their activities on the sidechain.
- $\mathsf{tx}_{\mathsf{swap}}$: Allows a client to submit a trade.
- $\mathsf{tx}_{\mathsf{mint}}$: Allows an LP to provide liquidity to a pool.
- $\mathsf{tx}_{\mathsf{collect}}$: Allows an LP to collect fees accrued due to providing liquidity.
- $\mathsf{tx}_{\mathsf{burn}}$: Allows an LP to withdraw her liquidity.
- $\mathsf{tx}_{\mathsf{Sync}}$: Allows a sidechain committee to sync the AMM base contract that resides on the mainchain.

$\mathsf{VerifyTx}(\mathsf{tx}) \to (0/1)$**:** Takes as input a transaction tx, and outputs 1 if tx is valid based on the syntax/semantics of its type, and 0 otherwise.

$\mathsf{VerifyBlock}(\mathcal{L}_{\mathsf{sc}}, \mathsf{B}_{\mathsf{btype}}) \to (0/1)$**:** Takes as input the current sidechain ledger state $\mathcal{L}_{\mathsf{sc}}$, a new block B with type $\mathsf{btype} = \mathsf{meta}$ or $\mathsf{btype} = \mathsf{summary}$. It outputs 1 if B is valid based on the syntax/semantics of the block type, and 0 otherwise.

$\mathsf{UpdateState}(\mathcal{L}_{\mathsf{sc}}, \mathsf{aux}, \mathsf{btype}) \to (\mathcal{L}'_{\mathsf{sc}})$**:** Takes as input the current sidechain state $\mathcal{L}_{\mathsf{sc}}$, and a set of pending transactions $\mathsf{aux} = \{\mathsf{tx}_i\}$ (if $\mathsf{btype} = \mathsf{meta}$) or $\perp$ (if $\mathsf{btype} = \mathsf{summary}$ since the inputs are the last epoch meta-blocks from $\mathcal{L}_{\mathsf{sc}}$). It reflects the changes induced by aux and outputs a new state $\mathcal{L}'_{\mathsf{sc}}$.

$\mathsf{Elect}(\mathcal{L}_{\mathsf{sc}}) \to (C, \mathsf{leader})$**:** Takes as input the current state of the sidechain ledger $\mathcal{L}_{\mathsf{sc}}$, and outputs an epoch committee C and its leader leader.

$\mathsf{Prune}(\mathcal{L}_{\mathsf{sc}}) \to (\mathcal{L}'_{\mathsf{sc}})$**:** Takes as input the current sidechain state $\mathcal{L}_{\mathsf{sc}}$, and produces an updated state $\mathcal{L}'_{\mathsf{sc}}$ in which all stale meta-blocks are dropped.

Note that $\mathsf{UpdateState}$ is the process of mining a new block on the sidechain based on its consensus protocol.

Security model. We aim to develop a secure state growth control solution that preserves the valid and secure operation of the underlying AMM. $\mathsf{ammBoost}$ builds a sidechain, which is basically a blockchain, that interacts with the application layer of the mainchain through the base AMM smart contract. This sidechain must be a secure ledger as defined below.

Ledger security. A ledger $\mathcal{L}$ is secure if it satisfies the following properties [45]:

Safety: For any two time rounds t_1 and t_2 such that $t_1 \leq t_2$, and any two honest parties P_1 and P_2, the confirmed state of $\mathcal{L}$ (which includes all blocks buried under at least k blocks, where k is the depth parameter) maintained by P_1 at t_1 is a prefix of the confirmed state of $\mathcal{L}$ maintained by party P_2 at time t_2 with overwhelming probability.

Liveness: If a valid transaction tx is broadcast at time round t, then with overwhelming probability it will be recorded on $\mathcal{L}$ at time at most $t + u$, where u is the liveness parameter.

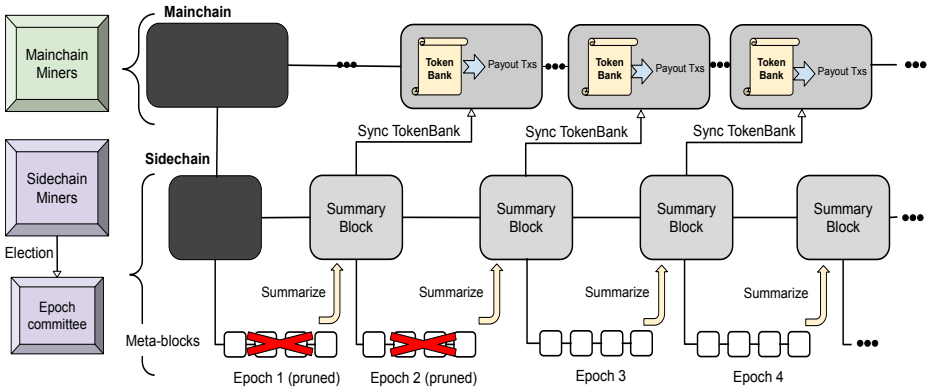

Fig. 2: The ammBoost framework (Txs is an abbreviation for transactions).

A ledger must record only valid transactions and blocks, thus its protocol is parameterized by predicates to verify transaction and block validity. For dApps, validity is governed by the code of their smart contracts, and miners ensure that the ledger state changes have been produced by a successful execution of this code. ammBoost reduces the AMM functionality deployed on the mainchain, and it processes most of the workload (following the same logic of the AMM) on the sidechain. Thus, in our security analysis, we show that ammBoost preserves the security and correct operation (i.e., safety and liveness) of the original AMM.

Adversary model. We assume the mainchain to be secure as defined above. For the sidechain, we have honest miners who follow the protocol, and malicious miners controlled by the adversary who may behave arbitrarily. The adversary can deploy new miners or corrupt existing ones, without going above the threshold of faulty nodes required by the sidechain consensus protocol. The adversary can see all messages and transactions sent in the system (since we deal with public permissionless blockchain systems) and can reorder these messages and delay them. We assume bounded-delay message delivery, so any sent message (or transaction) will be delivered within Δ time as in [48, 53, 59]. We assume slowly-adaptive adversaries [34] that can corrupt miners only at the beginning of each epoch. Lastly, we deal with PPT adversaries.

4 System Design

ammBoost changes the AMM deployment structure, as shown in Figure 2. The smart contract on the mainchain is minimal; it mainly tracks the token balances of the liquidity pools and the users, while most of the transaction processing is moved to the sidechain. Summaries of the sidechain processed traffic are used to sync the AMM smart contract on the mainchain. In this section, we present the design of ammBoost including system setup, architecture and operation, handling interruptions, and its security.

4.1 System Setup

The setup phase, as depicted in Figure 3, mainly specifies the traffic split between the chains and the summary rules for the sidechain traffic, as well as the sidechain parameters such as the epoch length and its consensus configuration (e.g., committee size).[7] Also, this phase involves deploying the AMM base smart contract on the mainchain and creating the sidechain.

Traffic classification and summary rules. AMM transactions and operations are divided into two groups: pool management and trading-related. Creating and managing token pools, as well as dispensing tokens to clients and LPs, are done on the mainchain since these deal with actual tokens. Flashes are also handled by the mainchain since they require instant token dispensing rather than at the end of the epoch (which is the case for any operation processed by the sidechain). The rest of the transactions, including swaps, mints, burns, and collects, are handled by the sidechain.

In ammBoost, the sidechain does not hold custody of tokens, it just tracks their balances based on the processed transactions. Thus, during each epoch, the sidechain produces two structures:

- A *payout* list containing users' public keys and the amount/type of tokens they should receive. This list is simply the updated deposit balance produced at the end of an epoch.
- A *liquidity position* list containing the position IDs, the public keys of their owners, balances, and any additional information needed by the liquidity management techniques, e.g., price ranges as in concentrated liquidity.

The actual token dispensing and deduction happen at the end of an epoch when the sidechain summaries are received. The new state of the pool token balances on the mainchain will be computed based on these lists. Also, the payout encompasses refunding any leftover in the deposits to their owners as will be shown shortly. In Section 4.2, we show the summary rules for each transaction type and how they contribute to the payout and payin lists.

Base smart contract TokenBank**.** The mainchain part of the AMM is a base smart contract called TokenBank. At an abstract level, as shown in Figure 4, this contract supports creating and managing token pools (i.e., tracking their balances and liquidity positions). It also provides the minimal interface needed to support users' activities on the sidechain, which is mainly creating deposits containing the tokens they want to trade or provide as liquidity. This is needed since the sidechain does not receive or send actual tokens, it only tracks balance evolution. Hence, a user deposits the total amount of tokens they would need during an epoch before this epoch starts, and TokenBank handles the payouts and payins produced by the sidechain when the epoch ends.

[7] The epoch duration impacts syncing frequency. Short epochs mean more sync-transactions, which incurs more gas cost and may impact throughput—as they are processed by the mainchain, however users would receive their tokens faster compared to long epochs. We empirically study the impact of this parameter in Section 6.

$\mathsf{SystemSetup}(1^\lambda, \mathcal{L}_{\mathsf{mc}})$: Takes as input the security parameter λ and the current mainchain state $\mathcal{L}_{\mathsf{mc}}$, and does the following:

1. Generate the sidechain configuration parameters:

- The epoch length ω.
- All parameters needed for the sidechain consensus protocol.
- Traffic classification rules.
- Summary rules and state variables.

2. Deploy the base contract TokenBank on the mainchain.

Outputs: epoch length ω, sidechain genesis block $\mathcal{L}_{\mathsf{sc}}^0$ (that references the block in the updated state $\mathcal{L}'_{\mathsf{mc}}$ containing TokenBank), and the address of TokenBank.

Fig. 3: System setup.

// ** State variables **
PoolSets: token-pair pools managed by the AMM.

Deposits: a map of users' public keys and the type/amount of tokens they deposited.

Positions: a map of users' public keys and the liquidity positions they own.

// ** Functions **
$\mathsf{createPool}(A, B)$: initializes a pool for the token pair (A, B).

$\mathsf{Deposit}(\mathsf{type}, \mathsf{amnt})$: allows a user to deposit an amount amnt of token with type type to be used for the next epoch.

$\mathsf{Sync}(\mathsf{aux})$: Sync the mainchain AMM state with the sidechain epoch summaries. The input aux contains the updated pool balances and liquidity positions, and the payin/payout lists.

$\mathsf{Flash}(\mathsf{aux})$: Receive a flash loan request where aux contains all required inputs, then calculate the token amount the pool can provide and initiate the callback process (more details can be found in Section 4.2).

Fig. 4: TokenBank abstract functionality.

As shown in Figure 3, system designers deploy TokenBank on the mainchain. Once the mainchain block containing this contract is confirmed, the genesis block of the sidechain $\mathcal{L}_{\mathsf{sc}}^0$ can be created such that it references this mainchain block.

Sidechain management. The sidechain in ammBoost is managed in a similar way as in chainBoost. At the beginning of each epoch, a committee from the sidechain miners is elected, which runs a PBFT consensus protocol to agree on mining meta/summary blocks and issuing sync-transactions. That is, the committee leader proposes new blocks or sync-transactions, and collect votes from the committee members. Once a vote majority is reached, the new block is added to the sidechain or the sync-transaction is sent to the mainchain. In ammBoost,

a sync-transaction is basically a call to the function Sync in TokenBank shown in Figure 4.

ammBoost differs from chainBoost in the aspect that the sidechain has its own miner population. In other words, mainchain miners are not responsible for managing the sidechain, and even may not know that a sidechain exists in the first place. Executing the TokenBank contract is like executing any other contract deployed at the application layer of the mainchain. As such, sidechain miners must possess some mining power to establish Sybil-resistant identities to be used in the committee election process. Any secure PBFT protocol in which election is based on the mining power can be used here, e.g., the proof-of-stake based protocol in [48].

4.2 System Operation—Transaction Processing

As mentioned before, ammBoost operates in epochs and rounds. The sidechain committee begins the epoch by retrieving the latest state, i.e., pool token balances, liquidity positions, and user deposits from the mainchain. It then processes all valid sidechain transactions, including swap, mint, burn, and collect (so users send these transactions to the sidechain). These are packaged into meta-blocks such that a meta-block is mined in each round. In the last round of the epoch, this committee produces a summary-block capturing the payouts for participating users, and any changes on liquidity positions, where the updated liquidity pool balances will be computed based on these lists. After that, it invokes the Sync function in TokenBank that resembles submitting a sync-transaction to update the AMM state on the mainchain, which is the state of TokenBank.

In this section, we describe how the various transactions are processed and summarized (Figure 5 captures how the sidechain workload is summarized in ammBoost). Before that, we want to point out that although a user will obtain her newly claimed tokens at the end of an epoch, once TokenBank is synced, they can use these tokens immediately during an epoch for trading. This is because new tokens will be added to the user deposit balance, and those that were used are deducted from deposit. Thus, the latest deposit state reflects the payout a user obtains (which includes refunding any deposit leftover).

Swaps. A swap transaction is a trade between the two tokens managed by a liquidity pool. A client provides an input of tokens and receives an output based on the price derived from the pool token balances and any user-defined trade conditions.

In order to execute a swap transaction, a user's deposit must cover the input token amount. An *exact input swap* transaction contains: the type and amount of input tokens to be traded, the minimum amount of output tokens the trade will accept (as a protection against slippage), a price limit that the trade should not exceed, and a deadline which is a round number after which the trade becomes

Input: meta-blocks $\mathsf{B}^1_{\mathsf{meta}}, \ldots, \mathsf{B}^n_{\mathsf{meta}}$ from an epoch and $\mathsf{Deposits}$ (the latter is the one retrieved from the mainchain at the beginning of the epoch).

Initialize: summary structures $\mathsf{sum}_{\mathsf{Payouts}}$ and $\mathsf{sum}_{\mathsf{Positions}}$.

for $i \in \{1, \ldots, n\}$ and every $\mathsf{tx} \in \mathsf{B}^i_{\mathsf{meta}}$ **do**
if $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{swap}}$ **then**
  $\mathsf{Deposits}[\mathsf{tx.userId}].\mathsf{amount}[\mathsf{in.type}] \mathrel{-}= \mathsf{tx.amount}_{\mathsf{in}}$
  $\mathsf{Deposits}[\mathsf{tx.userId}].\mathsf{amount}[\mathsf{out.type}] \mathrel{+}= \mathsf{tx.amount}_{\mathsf{out}}$
  Update fees in $\mathsf{sum}_{\mathsf{Positions}}$ for all positions used to fill tx
  // Liquidity amounts are computed as explained under
  //mints and burns.
elseif $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{mint}}$ **then**
  $\mathsf{sum}_{\mathsf{Positions}}[\mathsf{tx.posId}].\mathsf{amount}_A \mathrel{+}= \mathsf{tx.amount}_A$
  $\mathsf{sum}_{\mathsf{Positions}}[\mathsf{tx.posId}].\mathsf{amount}_B \mathrel{+}= \mathsf{tx.amount}_B$
  $\mathsf{sum}_{\mathsf{Positions}}[\mathsf{tx.posId}].\mathsf{priceRange} =$
    $(\mathsf{tx.lowerTick}, \mathsf{tx.upperTick})$
  $\mathsf{Deposits}[\mathsf{tx.userId}].\mathsf{amount}_A \mathrel{-}= \mathsf{tx.amount}_A$
  $\mathsf{Deposits}[\mathsf{tx.userId}].\mathsf{amount}_B \mathrel{+}= \mathsf{tx.amount}_B$
elseif $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{burn}}$ **then**
  $\mathsf{sum}_{\mathsf{Positions}}[\mathsf{tx.posId}].\mathsf{amount}_A \mathrel{-}= \mathsf{tx.amount}_A$
  $\mathsf{sum}_{\mathsf{Positions}}[\mathsf{tx.posId}].\mathsf{amount}_B \mathrel{-}= \mathsf{tx.amount}_B$
  $\mathsf{Deposits}[\mathsf{tx.userId}].\mathsf{amount}_A \mathrel{+}= \mathsf{tx.amount}_A$
  $\mathsf{Deposits}[\mathsf{tx.userId}].\mathsf{amount}_B \mathrel{+}= \mathsf{tx.amount}_B$
elseif $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{collect}}$ **then**
  $\mathsf{sum}_{\mathsf{Positions}}[\mathsf{tx.posId}].\mathsf{fees}_A \mathrel{-}= \mathsf{tx.amount}_A$
  $\mathsf{sum}_{\mathsf{Positions}}[\mathsf{tx.posId}].\mathsf{fees}_B \mathrel{-}= \mathsf{tx.amount}_B$
  $\mathsf{Deposits}[\mathsf{tx.userId}].\mathsf{amount}_A \mathrel{+}= \mathsf{tx.amount}_A$
  $\mathsf{Deposits}[\mathsf{tx.userId}].\mathsf{amount}_B \mathrel{+}= \mathsf{tx.amount}_B$
Output $\mathsf{sum}_{\mathsf{Payouts}} = \mathsf{Deposits}$, and $\mathsf{sum}_{\mathsf{Positions}}$

Fig. 5: Summary rules (userId is the user ID and posId is the liquidity position ID). Updated liquidity pool balances are computed by TokenBank (as part of processing Sync) based on the updated liquidity position and payout lists.

invalid if not executed by that time.[8] For an *exact output swap*, the goal is no longer to trade the exact amount of input tokens for the maximum amount of output tokens, but rather to minimize the amount of input tokens required to trade for the desired exact output. As such, the arguments of the function naturally change to reflect that, with the minimum output slippage protection changing to a maximum input slippage protection.

Processing. This is done using the original AMM logic for price balancing and output calculation. That is, $\mathsf{ammBoost}$ does not change the logic based on which an AMM operates, it just migrates that to the sidechain (this applies to the rest

[8] The recipient of the traded tokens is by default the issuer of the swap. This can be extended to support stating an explicit recipient that could be different from the issuer.

of the transactions as well). For an exact input swap, the sidechain committee computes the maximum amount of output tokens the user will receive for all of the input tokens provided. While for an exact output swap, the committee computes the minimum amount of input tokens needed to purchase the defined output. In both cases, these computations are based off the updated pool balance on the sidechain. In other words, as transactions are processed, the committee updates the pool state that was retrieved at the beginning of the epoch.

Furthermore, the fees for LPs whose liquidity was used in filling a swap will be computed. To elaborate, when a user submits a swap transaction, they pay a small additional fee, like 0.3% of their transaction's input or output value. It is paid in the token pair of the pool based on its net liquidity such that the token with the largest amount of net liquidity is used. For example, if a user provides 100 A input tokens in an exact input swap, and token A is the dominant token, then 0.3 A tokens are used for the LP fee, and 99.7 A tokens are used for the swap transaction. These fees are split up proportionally amongst the positions (based on the amount of liquidity they provide) that occupy the price range for which the swap was executed. ammBoost maintains a per-position fee balance, which is updated on every swap transaction, again using the same logic used by the underlying AMM to compute these fees.

Lastly, recall that a user will not get her actual traded tokens until the end of the epoch. However, she can use these tokens for trading on the sidechain since the sidechain tracks all balances. So basically, the deposit is a tuple of two values; one for each token type. When a swap is executed, the input token amount is deducted from the user's deposit while the output token amount is added to this deposit, thus allowing the user to use it immediately.

Summary rules. In the summary-block, the committee summarizes all swap transactions as follows: for every client, all her swaps are combined into a single tuple containing: the client public key, the total payout this user should receive. The latter encapsulates both a deduction from her deposit and a refund of any leftover in that deposit. For example, say a user started with a deposit of $(10A, 15B)$ and issued one swap during an epoch that traded 5 token A for 10 token B. The updated deposit (which will be the payout summary for that user) would be $(5A, 25B)$, which represents a payout of 10 B tokens, a deduction of 5 A tokens, and a refund of deposit leftover of $(5A, 15B)$. The same logic applies to the rest of the transactions.

Mints. Mint transactions allow the creation of new liquidity positions or modifying existing ones. An LP broadcasts a mint transaction to the sidechain that contains: the lower and upper ticks, representing the price range for which the liquidity is to reside, and the type/amount of the token to be used as liquidity. The mint will be accepted if the issuer LP's deposit (either the mainchain or sidechain one) can cover the provided liquidity amount.

Processing. This is also processed using the same logic used by the AMM. We resort to a simple approach to track ownership of positions; the sidechain committee generates a unique identifier (e.g., the hash of the mint transaction and the LP's public key) for a new position, and the owner is the public key of

the issuer LP. An existing position will receive an increase in its balance (or any other modifications on its price range) after verifying that the transaction issuer is indeed the rightful position owner. Mint transactions are initially invoked with a desired amount of token A and token B as input. The underlying AMM algorithms compute the maximum amount of liquidity (based off the input tokens) that the pool can take in at the current moment from both token types. These values represent the share of the pool liquidity now owned by the newly minted or modified position. The committee then deducts the provided liquidity amount (from both token types) from the corresponding LP's deposit balance.

Summary rules. All mint transactions are summarized as a list of liquidity positions with each position consisting of a tuple containing: the position identifier, the public key of the owner, the total amount of liquidity provided (or net change) for each token type under this position, and total amount of accrued fees. Note that the payin/payout of the LP is also updated when summarizing mint transactions; all provided liquidity token amounts are deducted from their deposits as shown in Figure 5.

Burns. A burn transaction allows a partial or complete liquidity withdrawal of a position. It is issued by an LP and contains: the position ID, the tick price limits, and the desired amounts of tokens A and B to be burned, and sent to the sidechain.

Processing. Processing a burn transaction boils down to determining if the issuer LP owns the position, then calculating the amount of liquidity this LP owns in a share, and converting that amount of liquidity into an amount of both tokens managed by the pool (using the original logic of the AMM). This would lead to updating the position range (upper and lower price ticks), or a deletion of the position if all its associated liquidity is withdrawn. If a deleted position has fees owed to it, the owner LP will receive these fees as part of her total payout computed at the end of the epoch, i.e., will be added to their deposit balance.

Summary rules. Burns are summarized as part of summarizing mint transactions detailed above. Burns adjust the net changes of the pool liquidity balance, i.e., they decrement this balance. Any fully withdrawn position will be removed from the TokenBank state. The withdrawn liquidity will be added to the LP's deposit balance to be reflected on the payout.

Collects. Collects allow LPs to collect the fees earned by their liquidity positions. An LP broadcasts a collect transaction containing the identifier of the position and fee amount to be collected.

Processing. This includes determining if the issuer LP owns the position, and checking if the amount they want to collect can be covered by their fee balance. If all is fine, the issuer LP's deposit is updated to reflect the amount of collected fees, and the fee balance for that position is adjusted accordingly.

Summary rules. Summarizing collect transactions is also part of summarizing mints/burns and the LP payout structure. That is, fee balance of the referenced position is decreased based on the collected amount, and the payout to the issuer LP is computed based on their updated deposit balance (to which the collected fee amount has been added).

Flashes. Flash transactions allow users to request short-term loans within the duration of one mainchain block. These are the only transaction type that ammBoost does not offload to the sidechain; the delay in paying out the actual tokens (which happens at the end of an epoch) limits the intended use of flash loans that span a very short period. As such, in ammBoost, flash transactions happen on the mainchain as in the original AMM architecture. Since flash loans take place in a singular block, they do not impact the pool balances; the amount of loaned tokens should be returned within one block period or the loan will be inverted. As a result, they do not invalidate any of the transactions processed on the sidechain based off the balance snapshot taken at the beginning of an epoch.

Remark 1. In terms of user experience, clients and LPs should be connected to both the mainchain and the sidechain, and their wallets should issue transactions to the destination chain based on the transaction type. For example, deposits should go the mainchain, while swaps/mints/burns/collects should go to the sidechain. Another difference is related to receiving the actual tokens, which are delayed until the end of the epoch in ammBoost. Since a user can use these immediately for trades within an epoch, the delayed payout has no impact. However, if a user wants to use these tokens on a different AMM, or wants to trade these with other token types to participate in another pool managing different token pairs, then they have to wait until the epoch. Still, overall, the delay is only one epoch.

4.3 System Operation—Chain Management

In this section, we discuss the syncing process, sidechain pruning, and how ammBoost recovers from interruptions.

Syncing TokenBank. The sidechain committee leader, after producing a summary-block containing all the summaries detailed earlier, calls the Sync function in TokenBank that resembles a sync-transaction submission. The inputs to this function call include: the list of payouts for all clients and LPs, and the list of liquidity positions with their updated information.

Authentication. TokenBank must ensure that the the Sync function invocation is issued by the rightful sidechain committee. We use a modified idea of quorum certificates (QC) [54, 72] combined with threshold signatures. In detail, to authenticate the Sync call for epoch $e+1$, the election of committee $e+1$ must happen during epoch e. Then, this committee runs a distributed key generation (DKG) [35] to generate a public verification key vk_c for the committee and secret shares of the signing key (one share per member) with a threshold of $2f+2$.[9] This committee initiates an agreement on vk_c, and then sends the agreement output to committee e along with proofs of election of each member who participated

[9] A committee size is $3f + 2$ and f is the maximum number of faulty nodes as in PBFT protocols, $2f + 2$ votes are needed to reach an agreement.

in the agreement.[10] Committee e verifies the election proofs, and then verifies that there is an agreement on vk_c. If everything is correct, committee e records vk_c on TokenBank by adding that to the Sync function call inputs they submit at the end of epoch e. During epoch $e+1$, committee $e+1$ runs an agreement over the Sync function call inputs and signs using their signing key shares, which result in one signature over these inputs. The leader then invokes Sync with the inputs and this signature. In turn, TokenBank verifies the signature using the recorded vk_c before accepting the summaries. By the security of the threshold signature scheme, this signature will be valid only if at least $2f+2$ committee members has signed.

Processing. If successfully verified, TokenBank processes the Sync function call by updating the list of positions based on the summaries, i.e., delete fully withdrawn positions, create new positions, or adjust existing ones, Then, it updates the pool balance based on the reported payouts and updated position list. Lastly, it dispenses the payouts to the referenced clients/LPs.

Sidechain pruning. ammBoost uses the *block suppression technique* from chainBoost. Once the transaction encapsulating the Sync function call is confirmed on the mainchain, all meta-blocks associated to this transaction will be pruned. The summary-blocks, as mentioned before, are permanent and represent checkpoints of the sidechain state in each epoch. So they can be used to verify the state of the AMM reflected by TokenBank state variables.

Handling interruptions. We identify the scenarios that can lead to operation interruption in ammBoost and how to recover from them. Recall that the sidechain committees use a PBFT-based consensus that assumes up to f of the elected miners can be malicious.[11] Thus, interruptions that could happen result from having a malicious or unresponsive leader. This leader may either propose an invalid meta/summary-blocks or invalid function call to Sync, or not initiate the agreement in the first place. Another interruption could result from rollbacks on the mainchain. That is, when the mainchain miners switch their canonical chain to the one satisfying a particular fork resolution criteria (i.e. the longest branch, or the heaviest one), causing the most recent blocks to be abandoned. This is an issue if the abandoned blocks contain Sync transactions.

Detection and recovery from these interruptions are done as in chainBoost [57], which we review briefly here. A leader that proposes an invalid block or Sync call can be easily detected by the committee when verifying this proposal. Once detected, the view-change technique [41] is used to elect a new leader. In the case of an unresponsive leader, if no agreement is initiated within a timeout period, a leader change is triggered. As for a leader that proposes invalid Sync inputs, a leader-change will not help since this happens at the end of an epoch when it is time for the new committee to take over. Thus, this case, and the rollback

[10] In our implementation, this election proof is the output of the verifiable random function (VRF) used in the election mechanism.

[11] This is valid under a committee size that guarantees satisfying this condition with overwhelming probability, where we adopt the committee size analysis from [57].

interruption, are addressed using the mass-syncing technique. The new committee issues a Sync call covering the summaries they produced in their epoch and those produced earlier in the impacted epochs.

Remark 2. A sidechain operates as a regular blockchain, thus any transactions that have not been processed in an epoch will be carried over to the epoch after. All sidechain miners receive transactions destined to the sidechain, but only the elected committee mines meta and summary blocks. Thus, when a new meta-block is mined, the committee and all other sidechain miners remove all published transactions in that block from their queues.

4.4 Security

Since ammBoost delegates the processing of the AMM transactions to the sidechain, and introduces pruning and state synchronization, we show that under this new architecture, the security and correct operation (i.e., safety and liveness) of the underlying AMM are preserved. In Appendix A, we prove the following theorem:

Theorem 1. ammBoost *preserves the safety and liveness of the underlying AMM.*

5 Implementation

To assess the performance gains that ammBoost provides for AMMs, we implement a proof-of-concept and conduct various experiments.[12] We chose a Uniswap-inspired use case to represent the underlying AMM. This section discusses the implementation details, while the experimental setup and results are discussed in the next section.

Sidechain implementation. For the sidechain, we use the chainBoost implementation from [3]. This code uses cryptographic sortition-inspired election mechanism [48] for the committee election, and this committee runs a BLS collective signing (CoSi)-based PBFT-based consensus algorithm [7]. We add the modifications needed for ammBoost; our sidechain has its own miners, and adopts the summary rules defined in Section 4. We also modify the syncing process to be an invocation to the Sync function in TokenBank authenticated using the threshold signature-based quorum certificate discussed earlier. We use a simplified version of the golang BLS library [68], and a pre-generated key to sign the Sync transaction. Furthermore, our sidechain implements two extra functions to aid in performing the AMM functionality:

- $\mathsf{CreateTx}_{\mathsf{Sync}}$: creates the Sync call inputs based on the summary-block, including the payouts, the updated liquidity positions, and the pool liquidity balance. This is called by the sidechain committee leader at the end of each epoch.

[12] We will open source our implementation.

– SnapshotBank: retrieves users' deposits at the beginning of an epoch.[13]

Mainchain details. For the mainchain, we utilize the Ethereum Sepolia testnet [19] using the hardhat development environment [14]. We implemented TokenBank in Solidity [20] and deployed it on Sepolia. In our implementation, the interfacing between the sidechain miners and the mainchain is handled through functionality provided by the Go-Ethereum project [13]. To allow TokenBank to authenticate the Sync function call, we implement BLS signature verification in solidity, where we use the 256-bit Barreto-Naehrig (BN256) curve operations defined in the Ethereum precompiles [40, 64]. We implement our hash-to-point functionality as the scalar multiplication of a Keccak256 hash of the Sync entries and the generator of the G_2 curve of BN256.

Use case: Uniswap-inspired AMM. We implement swaps, mints, burns, and collects using the same logic as in Uniswap (Appendix B). We do not implement flashes since they represent a very small portion of the traffic, and thus will not impact the performance gains we report. For simplicity, our use case implementation manages a single pool. We deployed two standard ERC20 contracts to provide the token pair traded in this pool and used in both the ammBoost and baseline experiments. Naturally, to test Uniswap V3 in an isolated environment, we use the UniswapV3Factory contract to deploy a new liquidity pool which hosts the two ERC20 tokens. In order to test against the baseline implementation of Uniswap V3, we wrote and deployed a smart contract to interface with the various Uniswap contracts as detailed in the Uniswap documentation [26]. This interface contract routes swaps to the swapRouter contract and mints/burns/-collects to the NFPM. Additionally, it manages all of the NFT liquidity positions (through the ERC721Reciever interface) created by the users in our experiments.

Traffic generation. Users generate traffic on both the mainchain and the sidechain. On the sidechain, the traffic follows the same distribution as in Uniswap (see Appendix C), i.e., 93.19% of the traffic is $\mathsf{tx}_{\mathsf{swap}}$, 2.14% is $\mathsf{tx}_{\mathsf{mint}}$, 2.38% is $\mathsf{tx}_{\mathsf{burn}}$ and 2.27% is $\mathsf{tx}_{\mathsf{collect}}$. Our implementation provides configuration settings to modify the distribution and volume of the generated transactions to test their impact on the reported performance metrics.

6 Performance Evaluation

6.1 Experiment Setup

We deploy our system on a computing cluster composed of 8 hypervisors, each running a 12-Core, 130 GiB RAM, VM, connected with 1 Gbps network link. This setup is capable of running around 8000 sidechain miners. Unless stated otherwise, an experiment length is 11 epochs, each of which consists of 30 sidechain rounds (a round lasts 7 sec). Our default meta-block size is 1 MB and a sidechain committee contains 500 miners. We deploy 100 AMM users, generating traffic

[13] ammBoost retrieves pool balances only for newly created pools, their updated balances can be computed by the sidechain based on the traffic it processes.

that arrives at a constant rate of $\rho = \lceil \frac{V_D \times b_t}{3600 \times 24} \rceil$ where $V_D = 25 \times 10^6$ is the chosen daily volume of transactions.

In reporting the results, we measure the following metrics:

1. *Throughput*: Number of transactions processed per second.
2. *Sidechain transaction latency*: The delay between a transaction submission and its appearance in a meta-block.[14]
3. *Mainchain transaction latency*: The delay between a transaction submission and its confirmation on Sepolia.
4. *Payout latency*: The delay between a transaction submission and the completion of TokenBank syncing in the epoch in which this transaction has been published on the sidechain. We measure this metric by reporting the sum of the sidechain transaction latency, the time needed to issue the Sync call, and the time needed to process the transaction encapsulating the Sync call on the mainchain.
5. *Gas cost*: The average number of gas units paid to process core transactions.
6. *Main and side chain growth*: The growth (in bytes) of both the main and side chains.

6.2 Comparison with the Baseline

We compare ammBoost against a baseline, which is a deployment of Uniswap V3 on Sepolia, as mentioned earlier.

On-chain (itemized) per-operation overhead. We evaluate the overhead of the deposit and the syncing processes in ammBoost, and compare that to the baseline Uniswap on-chain operations. We set the daily volume V_D to be 500K transactions (10x Uniswap). We use a Gas Profiler [23] to measure the gas cost of the different components of the Sync transaction. As shown in Table 1, we find that storing the state of the liquidity positions is the most expensive, as each consists of 192 bytes (or 6 words), incurring 22,100 gas units per word. The same gas cost of 22,100 gas unit per word is incurred when storing the liquidity pool balance. Each payout transaction incurs a constant fee of 15,771 units. A deposit of two tokens incurs a total cost of 105,392 units. The threshold signature-based quorum certificate incurs a fixed fee that corresponds to the gas cost of the BN256 operations needed for verification, and a fee that is proportional to the length of the summary data structure.

Overall, the gas cost of the Sync call is affected by the number of positions processed in an epoch, the number of deposits made for an epoch, and the updated pool liquidity balance; this cost does not scale with the number of processed transactions, but rather with the number of clients and liquidity providers. On the other hand, in baseline Uniswap the gas cost is proportional to the total generated traffic, where the numbers in Table 2 are per one transaction from each type. For the average latency, a Sync transaction does not depend on

[14] To obtain an accurate representation of this metric, and thus process a comparable amount of traffic, we empty the transaction queues after the end of each run.

Table 1: Mainchain latency and itemized gas cost for ammBoost operations ($|sum|$ is the size of the summaries).

Operation	**Sync**					**Deposit**
Module	Payout (each)	Storage (per 32 byte)	Authentication			(2 tokens)
			Hash To Point		Verify	
			Keccak256	ecMUL	Pairing	
Avg. gas	15,771	22,100	$30 + 6 \times \lceil \frac{\lvert sum \rvert}{256} \rceil$	6,000	113,000	105,392
MC. lat. (s)	15.28					54.60

Table 2: Mainchain latency and gas cost for Uniswap.

Operation	**Swap**	**Mint**	**Burn**	**Collect**
Avg. gas	160,601.45	435,609.86	158,473.43	163,743.04
MC. lat. (s)	31.34	42.24	12.72	13.45

Table 3: Operation storage overhead.

ammBoost Sync **component**	**Payout entry**	**Position entry**	vk_c	**Signature**
Size on Mainchain (B)	352	416	128	64
Size on Sidechain (B)	97	215		
Uniswap operation	**Swap**	**Mint**	**Burn**	**Collect**
Size on Mainchain (B)	365.27	565.55	280.21	150.18

any other mainchain transactions, so it is confirmed within one block on average. However, since a two-token deposit depends on two ERC20 approvals, and performs 2 transfers, it takes around 4 blocks in our experiments. The same behavior is observed in our Uniswap baseline, as a swap requires 1 approval from the user and a mint requires 2 approvals. Since a two-token deposit depends on two ERC20 approvals, and performs two transfers, it will take at least four rounds if the operations are done sequentially. For our Uniswap baseline, as a swap requires one approval, it takes a minimum of two rounds to be processed; a mint takes three rounds at least, at it requires two approvals.

We also report the per-operation storage cost. In particular, we report the cost breakdown for the Sync call on the mainchain and the summary-block size for ammBoost, and the transaction sizes for baseline Uniswap on the mainchain. For the Sync call, the sizes of payout and position entries vary greatly between the summarized changes in a summary-block and the Sync inputs submitted to the mainchain. This is due to the difference in encoding and binary packing between the side and main chains. On the mainchain, Ethereum's application binary interface (ABI) packing keeps track of the data and all the information needed to reinterpret it back, while on the sidechain we use simple binary packing. We also have an extra 6 words (192 bytes) storage overhead on the mainchain

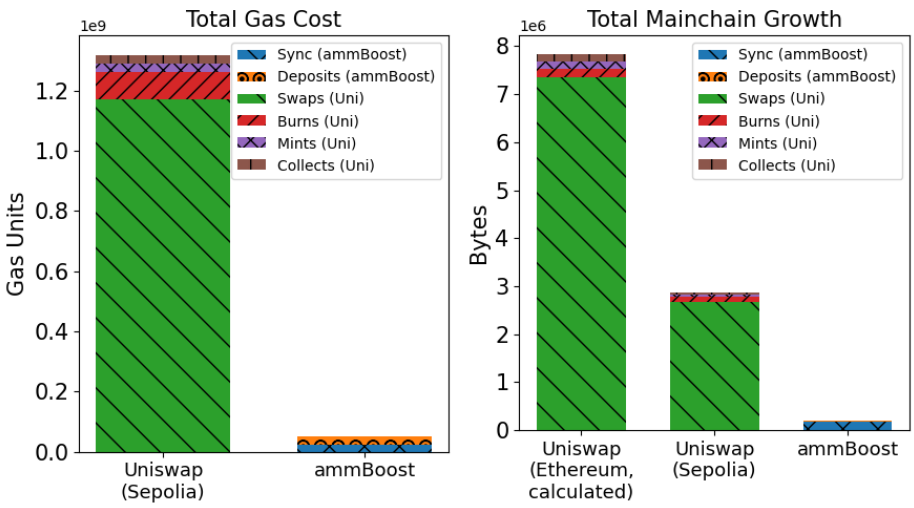

Fig. 6: Gas cost and chain growth comparison.

needed for the BLS signature and its public key (namely, vk_c) for authenticating the Sync call. We present our findings in Table 3.

As shown, for Uniswap, we notice that the transactions on Sepolia are smaller than the ones we observe on Ethereum (Appendix C). This is because these chains use different Uniswap transaction routers. The calls to the universal router used on Ethereum end up requiring more arguments, resulting in longer transactions. Uniswap on Sepolia deploys a simpler transaction router. Of note is that the simple router contract (Uniswap V3 router) and the more complex of the two (the Universal router) are both available on Ethereum, but the Universal router is not deployed to Sepolia.

In general, and as will discussed next, Uniswap incurs a larger storage cost as its transactions are quite large and all are logged on the mainchain. While for ammBoost, only (the less frequent) Sync call transaction is logged on the mainchain.

Overall comparison. We report the total gas cost and the mainchain state growth of the baseline Uniswap and ammBoost. We set the daily volume V_D to be 500K transactions (10x Uniswap) with the default traffic distribution. We measure the overall mainchain gas cost of relevant operations, and the state growth of the mainchain.

As shown in Figure 6, even if the sync transactions end up being heavy on gas as the number of positions and payouts increases, we achieve a 96.05% gas reduction when compared to Uniswap Sepolia. The high gas cost of the Sync transaction is offset by it being uncommon (one occurrence per epoch). On the other hand, the gas cost of swaps, mints, burns, and collects in Uniswap are high since all are processed on the mainchain (while in ammBoost these are processed on the sidechain). A similar trend is observed for the mainchain

Table 4: Scalability of ammBoost.

Daily volume	50K	500K	5M	25M
Throughput (tx/s)	0.42	3.41	33.04	138.06
Avg. sc latency (s)	7.13	7.13	7.13	231.52
Avg. payout latency (s)	120.71	120.71	120.71	346.49

state growth, where ammBoost provides 93.42% decrease in growth compared to Uniswap on Sepolia, and 97.60% decrease when compared to Uniswap on production Ethereum.[15]

6.3 Impact of Parameter Configuration

We study the impact of parameter configuration on ammBoost's performance, including traffic amount and distribution, block size, sidechain round duration, and number of rounds per epoch.

Scalability. In this experiment, we test the scalability of ammBoost (for a single pool) to understand its behavior under heavy traffic. We follow the same traffic distribution as in Uniswap and vary the daily volume $V_D \in \{50K, 500K, 5M, 25M\}$. We record the impact on throughput and transaction/payout latency as shown in Table 4.

Throughput-wise, we record a low throughput of 0.42 tx/s to 33.04 tx/s for a daily volume of 50K to 5M transactions (roughly 1x-100x Uniswap's daily volume). This is mainly due to the mainchain blocks not being full as this workload is way below the capacity that ammBoost can handle. While for traffic that is 500x Uniswap's daily volume, ammBoost achieves a throughput of 138.06 tx/s.

Latency-wise, we achieve a quasi-instant when the daily volume is between 50K and 5M (transactions that arrive at the beginning of the round get processed within the same round, while the residual amount of latency is due to transactions generated at the end of the epoch and processed in the next epoch). This leads to payouts being processed within one epoch (including the time needed to confirm the Sync transaction on the mainchain). Transaction congestion happens when the daily volume is 25M, resulting in higher average transaction and payout latency as the table shows.

Impact of block size. We test the impact of the sidechain block size with the goal of finding an optimal block size for our system. Thus, we compare different deployments of ammBoost with different block sizes against Uniswap's Sepolia deployment. We run the protocol with the following block sizes $\{0.5, 1, 1.5, 2\}$ MB, and we increase the daily volume to 50M transactions. We measure the impact on throughput and transaction/payout latency with the goal of identifying the block size that maximizes throughput while minimizing latency. Our results can be found in Table 5.

[15] The growth for Uniswap on production Ethereum is calculated by multiplying the count of each transaction type in our experiment by its size as reported in Appendix C

Table 5: Impact of different sidechain block sizes.

BlockSize (MB)	0.5	1	1.5	2
Throughput (tx/s)	68.97	138.61	207.52	276.43
Avg. sc. latency (s)	4357.00	1603.01	687.98	230.48
Avg. payout latency (s)	4472.63	1719.10	804.05	345.44

Table 6: Impact of different sidechain round durations.

Sc round duration (s)	7	11	16	21
Throughput (tx/s)	138.06	92.18	61.75	46.31
Avg. sc latency (s)	231.52	921.64	1950.92	2975.90
Payout latency (s)	346.49	1087.95	2193.85	3295.11

As expected, increasing the block size improves both throughput and latency, as more transactions can be packed in a block which reduces queue congestion. However, larger block sizes mean a larger propagation delay that could be problematic for short sidechain round duration. Thus, system designers should be careful when choosing an optimal sidechain block size, balancing between the block size that can handle the daily volume of transactions while capturing the intricacies of large network transfers.

Impact of sidechain round duration. Another important factor to study is the impact of the sidechain round duration. An ideal round duration should allow for consensus to conclude while maximizing throughput and minimizing latency. As PBFT agreement takes on average around 6 sec to conclude in our implementation, we test the following round duration values: 7, 11, 16, and 21 seconds, and report the performance metrics as before (Table 6).

Throughput-wise, we observe that as the block time increases, throughput decreases and latency increases. This is due to processing the same amount of transactions while increasing the time needed to produce a block. To choose an optimal block time, system designers need to take into consideration the time required for the sidechain consensus and network propagation delays, while aiming to generate new blocks as fast as possible.

Impact of the number of sidechain rounds per epoch. We test the impact of the number of sidechain rounds within an epoch. The goal is to find an epoch length that maximizes throughput, and minimizes transaction/payout latency, based on the optimal sidechain round duration from the experiment above. Thus, we pick our epoch to have $\{5, 10, 20, 30, 60, 96\}$ sidechain rounds, each of which lasts 7 sec, and report the performance metrics (Table 7).

Having short epochs negatively affects throughput and the sidechain latency. As a matter of fact, frequent summary-blocks harm performance since this leads to a larger number of Sync calls that are costly. At the same time, fewer transactions are processed within the epoch, thus affecting both latency and throughput. Longer epoch duration reduces the sidechain latency and increases throughput. However, this affects the payout latency adversely since Sync calls now are much

Table 7: Impact of number of sidechain rounds per epoch.

Epoch len (sc rounds)	5	10	20	30	60	96
Throughput (tx/s)	114.27	128.53	135.90	138.06	140.66	141.53
SC latency (s)	517.94	333.54	255.57	231.52	208.96	199.55
Payout latency (s)	545.12	337.86	334.81	346.49	434.94	546.04

Table 8: Impact of traffic distribution.

Swap %	60			80		
Mint %	20	10	10	10	5	5
Burn %	10	20	10	5	10	5
Collect %	10	10	20	5	5	10
Throughput (tx/s)	145.16	143.76	140.91	143.76	140.23	140.14
SC latency (s)	162.26	175.35	177.39	202.48	215.06	210.35
Payout latency (s)	277.99	291.05	293.03	317.23	329.81	324.43
Max sc growth (B)	31831	31831	31831	31831	31831	31831

fewer and spaced out (so users have to wait longer, as the epoch itself is longer, to obtain their actual token payouts). Also, this means that these users have to put larger deposits to cover their (long) epoch activities, which could be undesirable. Based on our results, we achieve the best payout latency when the epoch lasts for 20 sidechain rounds, which is equivalent to 140 sec.

Impact of traffic distribution. In this experiment, we evaluate different traffic distributions as follows (all the numbers are percentages): $(s, m, b, c) \in \{(60, 20, 10, 10), (60, 10, 20, 10), (60, 10, 10, 20), (80, 10, 5, 5), (80, 5, 10, 5), (80, 5, 5, 10)\}$, where (s, m, b, c) stand for swaps, mints, burns, and collects, respectively. As noted, in these configurations we keep the swap operations dominant to align with the baseline AMM traffic distribution observed in practice (see Appendix C). Our results can be found in Table 8.

When varying the traffic distribution, the metrics we report remain similar. This is because transaction sizes are very close, this yields blocks containing approximately the same number of transactions, regardless of the transaction distribution. As for the maximum chain growth, it is bounded by the number of users participating during an epoch and the number of positions they create. Thus, it remains invariant even with a variation of transaction distributions since the number of users is the same.

7 Conclusion

We presented ammBoost, a secure state growth controller and throughput booster for AMMs. It combines a dependent-sidechain architecture with a functionality split of the AMM. The AMM is divided into two parts: a base smart contract called TokenBank residing on the mainchain, which manages token pools, users' deposits and payouts, and any transaction type that must be handled by the mainchain. And a sidechain part that process most of the workload. ammBoost introduces several techniques to address challenges resulting from the unidirectional dependency between the mainchain and the sidechain. We analyze the security of our system and conduct thorough performance evaluation experiments. The results show the great potential of ammBoost in reducing the on-chain storage footprint of AMMs and boosting their scalability.

Acknowledgments

The work of M.E.N. is supported by NSF under Grant No. CNS-2226932, and the work of G.A. is supported by the Latest in DeFi Research (TLDR) fellowship funded by Uniswap Foundation.

References

1. Amms market share. https://www.coingecko.com/research/publications/centralized-crypto-exchanges-market-share.
2. Btcrelay. http://btcrelay.org/.
3. chainboost source code. https://github.com/CSSL-UConn/chainboost-release.
4. Chainstack. https://chainstack.com/.
5. The cheater checking problem: Why the verifier's dilemma is harder than you think. https://medium.com/offchainlabs/the-cheater-checking-problem-why-the-verifiers-dilemma-is-harder-than-you-think-9c7156505ca1.
6. Cosmos. https://cosmos.network/.
7. cothority/blscosi. https://github.com/dedis/cothority/tree/main/blscosi/blscosi.
8. Curve amm. https://curve.fi/.
9. Dodo dex. https://dodoex.io/en.
10. Dune query to retrieve the number of each transaction type per year. https://dune.com/queries/3591431/6049916/92e6972f-2f75-42dc-bee9-bcf28fb46afe.
11. Ethereum blockchain explorer. https://etherscan.io/txs.
12. Filecoin. https://filecoin.io/.
13. Go-ethereum docs. https://geth.ethereum.org/docs.
14. The hardhat ethereum development environment. https://hardhat.org/.
15. Livepeer. https://livepeer.com/.
16. Optimism. https://www.optimism.io/.
17. Optimistic rollup is not secure enough than you think — game theoretic approach for more verifiable rollup. https://medium.com/onther-tech/optimistic-rollup-is-not-secure-enough-than-you-think-cb23e6e6f11c.
18. Polygon. https://polygon.technology.

19. Sepolia ethereum testnet. https://sepolia.etherscan.io/.
20. Solidity scripting language. https://docs.soliditylang.org/en/v0.7.4/.
21. Starkware solutions. https://starkware.co/.
22. Sushiswap. https://www.sushi.com/swap.
23. Tenderly — full-stack web3 infrastructure. https://tenderly.co/.
24. Top automated market maker (amm) coins today by market cap. https://www.forbes.com/digital-assets/categories/automated-market-maker-amm/?sh=3488af897b18.
25. Uniswap documentation. https://docs.uniswap.org/contracts/v3/overview.
26. Uniswap pool interaction guide. https://docs.uniswap.org/contracts/v3/guides/providing-liquidity/the-full-contract.
27. Uniswap protocol. https://uniswap.org/.
28. Uniswap reference implementation. https://github.com/Uniswap.
29. Uniswapv3subgraph. https://thegraph.com/hosted-service/subgraph/uniswap/uniswap-v3.
30. Xdai. https://www.xdaichain.com/.
31. zksync. https://zksync.io/.
32. Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 whitepaper. 2021.
33. Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contract platform. In *NDSS*, 2018.
34. Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. Divide and scale: Formalization of distributed ledger sharding protocols. *arXiv preprint arXiv:1910.10434*, 2019.
35. Renas Bacho and Julian Loss. On the adaptive security of the threshold bls signature scheme. In *ACM CCS*, 2022.
36. Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. 2014.
37. Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. Maximizing extractable value from automated market makers. In *Financial Cryptography and Data Security*, 2022.
38. Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. *IACR Cryptol. ePrint Arch.*, 2020.
39. Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *IEEE S&P*, 2020.
40. Vitalik Buterin and Christian Reitwiessner. Eip-197: Eip-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128, 2018. https://eips.ethereum.org/EIPS/eip-197.
41. Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *Usenix OsDI*, 1999.
42. George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In *NDSS*, 2016.
43. Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, 2015.
44. Robin Fritsch. Concentrated liquidity in automated market makers. In *ACM CCS Workshop on Decentralized Finance and Security*, 2021.
45. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015.

46. Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov. Zendoo: A zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. In *IEEE ICDCS*, 2020.
47. Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *IEEE S&P*, 2019.
48. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *ACM SOSP*, 2017.
49. Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security*, 2018.
50. Rami Khalil, Arthur Gervais, and Guillaume Felley. Tex-a securely scalable trustless exchange. *Cryptology ePrint Archive*, 2019.
51. Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*, 2019.
52. Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *Usenix Security*, 2016.
53. Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE S&P*, 2018.
54. Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11):1–11, 2014.
55. Sergio Lerner. Drivechains, sidechains, and hybrid 2-way peg designs. 2016. `https://docs.rsk.co/Drivechains_Sidechains_and_Hybrid_2-way_peg_Designs_R9.pdf`.
56. Jiasun Li. On the security of optimistic blockchain mechanisms. *Available at SSRN 4499357*, 2023.
57. Zahra Motaqy, Mohamed Najd, and Ghada Almashaqbeh. chainboost: A secure performance booster for blockchain-based resource markets. In *EuroS&P*, 2024.
58. Andreas Park. The conceptual flaws of constant product automated market making. *Available at SSRN 3805750*, 2021.
59. Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, 2017.
60. George Pîrlea, 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.
61. Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, pages 1–47, 2017.
62. Mohsen Pourpouneh, Kurt Nielsen, et al. Automated market makers for cross-chain defi and sharded blockchains. *arXiv preprint arXiv:2309.14290*, 2023.
63. Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *IEEE S&P*, 2022.
64. Christian Reitwiessner. Eip-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128, 2017. https://eips.ethereum.org/EIPS/eip-196.
65. Ionut Rosca, Alexandra-Ina Butnaru, and Emil Simion. Security of ethereum layer 2s. *Cryptology ePrint Archive*, 2023.
66. Yuechen Tao, Bo Li, Jingjie Jiang, Hok Chu Ng, Cong Wang, and Baochun Li. On sharding open blockchains with smart contracts. In *IEEE 36th International Conference on Data Engineering (ICDE)*, 2020.

67. Junfeng Tian, Hongwei Xu, and Jin Tian. Slchain: A secure and low-storage pressure sharding blockchain. *Concurrency and Computation: Practice and Experience*, 36(3):e7918, 2024.
68. Bjorn van der Laan. GitHub - BjornvdLaan/BGRVerify — github.com. `https://github.com/BjornvdLaan/BGRVerify`. [Accessed 21-05-2024].
69. Dabao Wang, Siwei Wu, Ziling Lin, Lei Wu, Xingliang Yuan, Yajin Zhou, Haoyu Wang, and Kui Ren. Towards a first step to understand flash loan and its applications in defi ecosystem. In *the International Workshop on Security in Blockchain and Cloud Computing*, 2021.
70. Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2016.
71. Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Computing Surveys*, 55(11):1–50, 2023.
72. Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019.
73. Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Horizontal scaling blockchain via full sharding. In *ACM CCS*, 2018.

A Security Analysis

Informally, ammBoost preserves the correct behavior of the AMM since it processes the workload using the same logic adopted by the AMM itself. Also, since the sidechain adopts a secure PBFT consensus protocol, with the assumption that only up to f miners in any epoch elected committee can be malicious, the committee only agrees on valid records that conform with the AMM operation and rules. Furthermore, since the AMM is a dApp deployed on top of a secure smart contract-enabled blockchain (i.e., the mainchain), ammBoost will not impact the liveness and safety of other applications deployed on this chain or the security of the mainchain. Thus, we focus on the safety and liveness of the AMM, i.e., honest miners agree on the confirmed state of the AMM contract, the AMM state grows over time meaning that its workload is being processed, and that the produced state changes are valid based on the AMM logic.

In order to prove Theorem 1, we prove two lemmas showing that ammBoost preserves safety and liveness of the underlying AMM (these proofs are inspired by those found in [57]).

Lemma 1. ammBoost *preserves the safety of the underlying AMM.*

Proof. Since ammBoost implements meta-block pruning and mainchain (i.e., TokenBank) state syncing, we identify the following threats that may impact safety in our system:

– Invalid processing of AMM transactions: the sidechain committee does not follow the AMM logic in processing transactions, or accept transactions from users who do not own enough deposits on the mainchain, or process these transactions based off an invalid initial state of the token pool.

– Out-of-sync AMM state on the mainchain: A committee leader does not issue a Sync function call at the end of the epoch, causing the state of the AMM on the mainchain and the state maintained on the sidechain to be out of sync. This also could happen due to rollbacks on the mainchain causing recent Sync calls to be lost.
– Invalid syncing: A sidechain committee agrees on invalid inputs (or syncing information) for the Sync function call, or an illegitimate committee pretends to be the elected one and issues such an invalid sync.
– Violating sidechain quality: A sidechain committee mines invalid meta- and summary-blocks.

We show how ammBoost mitigates these threats, which mainly relies on the use of a secure PBFT consensus with a secure committee election mechanism, a secure threshold digital signature scheme, and the leader-change mechanism to handle the case of malicious/unresponsive committee leader.

Invalid processing of AMM transactions. The sidechain in ammBoost uses the same logic adopted by the underlying AMM to process all transactions. This is based off the latest state of the AMM (with respect to user deposits, liquidity positions, and pool balances from TokenBank on the mainchain). The use of a secure PPFT protocol, with a committee size that guarantees having at maximum f malicious parties (among a committee of size $3f + 2$) with overwhelming probability as mentioned earlier, guarantees that only valid records that conform with these rules will be accepted in meta-blocks. Also, this guarantees that summary-blocks are also valid based on the summary rules in ammBoost and these meta-blocks. Meta-blocks do not get pruned until their Sync call transaction is confirmed on the mainchain, so anyone can verify the validity of these blocks and the validity of the summary-block (as well as the Sync call).

Out-of-sync AMM state on the mainchain. This is mitigated by the mass-syncing process. An unresponsive leader who does not initiate an agreement on the Sync function call, or does not submit the result of the agreement to TokenBank, is easily detected by the new committee as no function call has been issued and recorded on the mainchain. The new committee then syncs TokenBank based on the summaries it produces in its epoch and the ones in the previous (one or multiple) epoch. Same for any rollbacks that may happen on the mainchain, mass-syncing will include all summaries that has been lost due to the rollback.

Invalid syncing. If a leader issues invalid syncing information, the committee (which has honest majority) will not endorse these inputs. So simply this invalid syncing information will be ignored, and mass-syncing (discussed above) will handle the syncing within the next epoch. Having an illegitimate committee pretend to be the rightful one to submit (invalid) sync is addressed in ammBoost using the syncing authentication mechanism discussed in Section 4.2. The committee of epoch e will not accept the generated committee public verification key vk_c unless there are valid proofs of election showing that the newly claimed committee is the rightful one (so this relies on the security of the election mechanism to be publicly verifiable). Furthermore, an illegitimate committee (or an attacker) instead may try to forge a signature over the syncing information un-

der a valid vk_c, which succeeds with negligible probability by the security of the used threshold digital signature scheme.

Violating sidechain quality. As mentioned earlier, this could happen if the leader proposes invalid meta- or summary-blocks and the committee agrees on that. Agreement will not happen since we assume a secure PBFT-based consensus protocol. Also, a malicious leader who may propose invalid blocks will be detected by the committee when verifying the blocks. In this case, they will reject the proposal and initiate a leader-change to elect a new leader who will take over for the rest of the epoch.

Accordingly, ammBoost satisfies the safety of the AMM and will not result in any correctness or security threats.

Lemma 2. ammBoost *preserves the liveness of the underlying AMM.*

Proof. The liveness of the sidechain impacts the liveness of the AMM in the sense that any liveness threats on the sidechain will impact the operation progress of the AMM. Note that we assume the mainchain to be secure; since the AMM is a dApp deployed at the application layer of this chain, the mainchain liveness is not impacted in any case.

We identify the following threats that may arise and violate the liveness of the AMM under our setting:

- Denial of service (DoS) attacks: The sidechain committee deliberately ignores and omits transactions coming from particular clients or LPs.
- Violating sidechain liveness: the sidechain committee does not mine meta- and summary-blocks or does not submit a Sync function call in a timely manner (that could be due to malicious/unresponsive leader or malicious/unresponsive committee that does not reach an agreement).
- Violating the public verifiability of the sidechain: this covers all threats related to the syncing and pruning of meta-blocks that may impact the public verifiability of the sidechain (which in turn impacts the public verifiability of the AMM).

Proving that ammBoost mitigates these threats is the same as in [57]. For completeness, we review that proof arguments here. *DoS* is addressed by having a rotating committee election (a new committee is elected for each new epoch) such that this committee has an honest majority. A leader that targets particular users, and so omit their transactions from all proposed meta-blocks, will operate for one epoch and then a new committee with a new leader will be elected for the next epoch. Thus, maintaining a situation where all future leaders are malicious and perform the same DoS is unfeasible. *Sidechain liveness* is satisfied due to the use of a secure PBFT consensus and having a large enough committee that satisfies honest majority (so they will be active during the agreement). Also, leader-change allows changing a leader who deliberately attempt to stall the network by not proposing new blocks, and mass-syncing will address the case of a malicious leader who does not initiate agreement on a summary-block or Sync function call. *Public verifiability* is guaranteed by the security of the

PBFT consensus; meta-blocks are not pruned until their corresponding Sync call is confirmed, and summary-blocks are permanent. Also, by having the AMM base smart contract TokenBank on the mainchain synced correctly based on the sidechain summaries (as discussed in the proof of Lemma 1), summaries are not lost. All of these allow anyone to verify the validity of the evolving state of the AMM. Accordingly, ammBoost preserves the liveness of the AMM.

B Concrete Usecase: Uniswap

Uniswap has three versions: Uniswap V1, released in November 2018, consisted of the baseline protocol that implemented ERC20 token swaps with Ethereum and all of the liquidity management methods (mint, burn, collect). Uniswap V2, released in August 2020, introduced ERC20 to ERC20 swaps, liquidity provision incentives, and oracles. And Uniswap V3 released in May 2021, introduced concentrated liquidity, a nonfungible representation of liquidity positions, and further improvements to the oracle systems. Uniswap adopts the constant product formula for computing the trading price described in Section 2. Uniswap is among the most popular AMMs in practice and commands a large market share in the AMM industry. In this section, we provide an overview of the set of contracts that implement the Uniswap functionality on Ethereum, and the execution trace of the supported transactions.

B.1 Uniswap Supporting Contracts

Based on the Uniswap documentation [25] and its reference implementation [28], Uniswap is implemented as a set of five contracts: PoolDeployer, PoolFactory, NonfungiblePositionManager, NonfungibleTokenPositionDescriptor, and SwapRouter.

Pool factory and deployer. The PoolFactory and PoolDeployer contracts are responsible for setting up new token pools. The pool deployer contract provides the interface, and the pool factory contract creates the actual pool. Once a pool is created, clients and LPs can start interacting with the pool.

Nonfungible position manager and token descriptor. These contracts manage the liquidity positions by handling processes associated with minting, collecting, and burning/adjustment of liquidity positions. The NonfungiblePositionManager contract serves as a "pit stop" for an LP's input tokens, such that the LP deposits input tokens for mint transactions before executing the mint functionality of a particular pool. This intermediate step allows Uniswap to guarantee that the input tokens will actually be delivered by the LP, as they are deposited in the first step, and automatically retrieved from the NonfungiblePositionManager contract by the pool contract when needed. The LP can later retrieve any tokens not used by the mint transaction. These two contracts also implement a unique NFT-based identifier for liquidity positions such that LPs can trade positions amongst themselves.

Swap router. The SwapRouter contract manages the swapping process. It implements functions such as ExactInput and ExactOutput to facilitate specific kinds of swaps. The SwapRouter also serves as a "pit stop" for input tokens, requiring clients to deposit tokens they want to trade before performing swap transactions.

There are additional smart contracts deployed in the Uniswap ecosystem, e.g., lens contracts which act as an on-chain oracle to record the price and liquidity history of a given pool. We do not provide further information about such contracts since we focus on the core functionality of Uniswap in our usecase implementation.

B.2 Transaction Execution Trace

The core transaction types supported in Uniswap, namely, swaps, mints, burns, collects, and flashes, are executed as follows.

Swap. Regardless of the type of swap (exact in/out) being executed, clients must first deposit their input tokens in the SwapRouter contract and approve it to spend their tokens. The client then calls the relevant function of the SwapRouter contract (ExactInput or ExactOutput) to submit a swap transaction. If the user is performing an ExactOutput swap, they should implement an additional set of conditional transfers to occur after the call to ExactOutput to retrieve unspent input tokens.

Internally to either function, the pool's swap function is called (Swap_{pool}). Swap_{pool} determines the price of the swap, distributes the liquidity provider fee across the positions who's liquidity is used to fill in the swap, and transfers the output tokens to the client before invoking SwapCallback. The SwapCallback function is called to retrieve the necessary amount of input tokens from the NonfungiblePositionManager contract. The client's contract can now call any additional transfers to retrieve unspent input tokens (in the case of ExactOutput).

Mint. The user first creates a smart contract capable of receiving ERC721 tokens. This contract must implement the following: a method to receive and store the nonfungible position tokens, and another method to execute the mint. Alternatively. the user may simply forgo the ERC721 receiver contract, allowing their NFT positions to remain as part of the NFTPM contract. Should a user decide to do this, they can simply invoke the same relevant functions of the NFTPM below by using any library which allows interfacing with smart contracts. Mint execution encompasses the following:

- The LP transfers their input tokens to the NonfungiblePositionManager contract, and authorizes it to spend their tokens when executing the mint.
- The LP calls the NonfungiblePositionManager contract's mint method (Mint_{NFPM}), which creates the NFT position structure, and then calls addLiquidity to create the liquidity position.

- The addLiquidity function retrieves the relevant pool from the passed tokens and fee tier. Using the current price ratio, and the desired amount of tokens to be added as liquidity (passed by the user), an applicable liquidity share is calculated using the function getLiquidityForAmounts. Then, the pool's mint function Mint_{pool} is called, passing the liquidity value computed by getLiquidityForAmounts.
- Mint_{pool} creates a new LP with the liquidity share provided, and outputs the exact amount of the token pair required for the position. Then it calls the MintCallback function, after which the amount of each token to be added to the pool is returned to the NonfungiblePositionManager contract.
- The MintCallback function does the following: it verifies that the caller is a valid pool contract, and then transfers the used input tokens from the NonfungiblePositionManager contract to the pool contract.
- Mint_{NFPM} returns the nonfungible position token, as well as the amount of each token actually added to the pool.
- The LP can retrieve any unspent input tokens from the NonfungiblePositionManager contract; this is why the LP needs to implement a method to receive and store an ERC721 token in their contract.

Collect. To execute a collect transaction, the LP calls NonfungiblePositionManager's collect method ($\mathsf{Collect}_{NFPM}$), passing the amount of fees they wish to withdraw along with the nonfungible position token's ID representing their liquidity position. The function $\mathsf{Collect}_{NFPM}$ verifies that the transaction issuer is indeed the position owner, and then identifies the target pool based on the token ID. After that, it retrieves the current token amount owed to the owner from the position through the fee calculation process. The latter is an optimization introduced in Uniswap V3 to accommodate for concentrated liquidity positions and to reduce the overall gas usage. Specific details on the calculation process for fees in Uniswap V3 can be found in its whitepaper [32].

Burn. To burn a position, the following steps take place:

- The LP withdraws all the tokens owned by that position. To do so they first call decreaseLiquidity a method of the NonfungiblePositionManager contract. This function retrieves the relevant pool contract, and calls the pools Burn function.
- The burn function takes the requested amount of tokens to burn, calculates the actual share of liquidity owned by the position which can be burnt (up to the requested amount), and decrements the calculated amount from the positions owned tokens. Finally, it adds the decremented amount to the liquidity positions owed-tokens metric, such that they can be withdrawn by invoking collect.
- Once the LP have decreased the liquidity owned by their position to zero, they can invoke collect to retrieve those funds before calling Burn_{NFPM}.
- Burn_{NFPM} checks that the passed liquidity position does not own any shares of liquidity and all owed tokens have been collected. Should these checks pass it deletes the liquidity position and the NFT associated with it.

Flash. To execute a flash transaction, the client begins by writing and deploying a smart contract which overwrites the $\mathsf{Flash}_{callback}$ method of the liquidity pool. The $\mathsf{Flash}_{callback}$ function is responsible for paying back the loan. As such, should a client want to perform arbitrage with the loan, they begin by overwriting $\mathsf{Flash}_{callback}$, simply adding in solidity code to execute their arbitrage opportunity. They can then call the flash function of the liquidity pool from which they would like to execute a flash transaction. Flash itself simply transfers the requested loan of tokens to the client, where they are used for the arbitrage opportunity, before being re-transferred to the pool, plus the associated fees, by $\mathsf{Flash}_{callback}$. Should the arbitrage prove non-profitable, or the contract fail to pay back the flash loan for any reason, the entire transaction is concluded, resulting in the pool never having transferred the loan in the first place. This is possible due to the entire flash process occurring in a single Ethereum transaction.

Remark 3 (On NFT-based liquidity positions). Uniswap V3 introduced an NFT-based approach, using an ERC721 wrapper, to track ownership of liquidity positions. This approach allows for a streamlined process for the verification and transfer of ownership of a position. This can be also adopted in ammBoost. At a high level, TokenBank can be extended to support the NFT approach by utilizing the same implementation found in Uniswap. The caveat though is that creating an NFT will wait until the end of the epoch since it requires mainchain operation (now in ammBoost positions can be created immediately and synced back to the mainchain since tracking owenership relies on the LPs' public keys and identifiers are generated at random). Thus, any operations on these new positions has to wait until the next epoch after creating the position NFT.

C Uniswap Traffic Analysis

In order to find the volume of each transaction type, we used the following query on Dune analytics uniswap_v3_ethereum dataset (the following citation is a direct link to the query used [10]). The query retrieves and counts all of the transactions happened since 2019, splitting them by year and transaction type. uniswap_v3_ethereum is one of Dune analytics "decoded projects", meaning that it is a dataset formed from the ABI of the smart contracts that operate the protocol in question. Once a user submits a contract for decoding, Dune uses the ABI to generate a table of transactions that is query-able by function call or event. As such, the uniswap_v3_ethereum is a set of tables which contain the decoded smart contracts that constitute the Uniswap V3 protocol. Since we are interested in transaction volumes, the above query counts all transactions since 2019 by pulling any row in the tables uniswap_v3_ethereum.Pair_call_burn, uniswap_v3_ethereum.Pair_call_collect, etc., for which the
call_block_time is $\geq$ 01/01/2019.

Traffic distribution or transaction type frequency. We calculate the frequency of each transaction type by computing the number of transactions of that particular type divided by the total number of transactions from all types. The

Table 9: Transaction type breakdown in Uniswap traffic for the year 2023.

Transaction Type	Percent of all traffic	Volume per 24 hr	Average Size (B)
Swap	93.19 %	52,379	1007.83
Mint	2.14 %	1,204	814.49
Burn	2.38 %	1,338	907.07
Collect	2.27 %	1,275	921.80

volume of transactions is gathered by the query above. The frequency shown in Table 9 is calculated for the year 2023.

Number of trades per 24 hours by transaction type (volume). We calculate this metric by taking the 2023 yearly total transaction count (found by the query above), and then compute the average daily volume of each transaction based on the frequency computed above. The results can be found in Table 9.

Transaction sizes. In order to collect the average size of each transaction type, we implemented a python script to interact with an Ethereum node hosted by chainstack [4]. We first collected the transaction hashes of a sufficient amount (approximately 40,000 swaps and 10,000 of all other transaction types) of each transaction type from the Uniswap V3 subgraph [29] (this can similarly be done with the dune query provided above, by modifying the select on line 37 to include the transaction hash). Then, we ran a script to analyze the collected transactions. This script basically iterates through the json file which contains our aggregated transaction hashes. For each transaction hash, it performs a web3.eth.get_raw_transaction query to retrieve the full raw transaction size. After that, it computes the average size for each transaction type (which is basically sum of total size divided by the number of transactions). The results are also found in Table 9.