SOLAYER CHAIN: Infinitely Scalable Hardware-Accelerated SVM Network

Jason Li and Chaofan Shou

(lxj, shou)@berkeley.edu

January 6, 2025

Abstract

We introduce SOLAYER CHAIN, a next-generation blockchain architecture that scales a single, global state machine to achieve unprecedented throughput, low latency, and robust composability. Unlike conventional vertical scaling or sharded rollups, SOLAYER CHAIN maintains atomic state transitions by distributing the workload across microservices and specialized hardware accelerators. It offloads signature checks, transaction filtering, pre-execution simulation, scheduling, and storage to distinct scalable clusters, each optimized for its function. Transactions are speculatively executed before reaching the sequencer, then finalized through a hybrid Proof-of-Authority-and-Stake consensus. The architecture leverages Infiniband RDMA for near-microsecond inter-node communication and advanced concurrency control strategies to minimize re-execution. A shared-nothing key-value store, sharded across multiple nodes, supports arbitrarily large account data with load balancing. SOLAYER CHAIN pushes blockchain performance to hardware limits, targeting 1M+ TPS and 100Gbps+ network bandwidth. This design provides a path for next-generation applications that require high throughput, low fees, and a seamless composable environment. SOLAYER CHAIN also introduces multiple user experience improvements. Hooks allow developers to embed post-transaction logic—such as arbitrage, liquidations, and accounting—directly within the chain, while jumbo transactions, cross-chain calls, and built-in OAuth support further enhance developer and user experience.

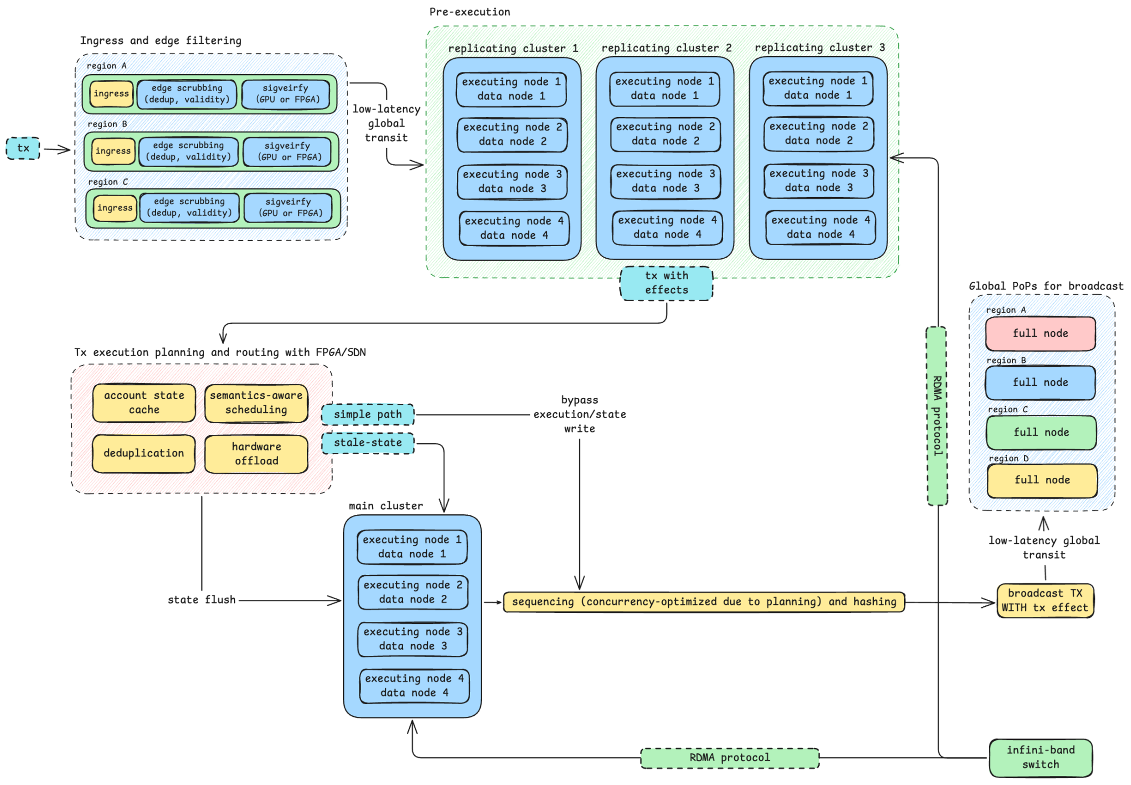
1 Introduction

Blockchain scalability remains a foundational challenge for decentralized systems, as rising transaction volumes demand faster processing, larger data storage, and robust consensus protocols—all without compromising security or decentralization. High-performance blockchains can reduce transaction times and fees, foster a positive user experience, and open the door to large-scale applications.

As the industry matures, the demand for block space continues to increase. On platforms like Ethereum, heightened demand raises gas fees due to fee auctions; on Solana, a flooded network can lead to dropped or expired transactions. Typical metrics such as transactions per second (TPS) shed light on throughput, yet blockchains have grown beyond simple transfers to support Turing-complete smart contracts and complex functionality. Meeting these demands requires rethinking how block space is provisioned and how state is managed.

Horizontal scaling approaches, such as Ethereum's Layer 2 (L2) rollups, move transaction processing off the main chain into separate, independently updated states, posting final results back on L1. While this boosts throughput, it fragments a once-unified global state, weakening composability, atomic state transitions, and liquidity. For instance, if an asset is split across multiple pools on different rollups, a cross-rollup swap can suffer higher slippage relative to a single, unified liquidity pool. Vertical scaling strategies, exemplified by Solana, aim to maximize throughput within a single-state machine by optimizing the virtual machine for parallel execution, writing high-performance software, and running on powerful consumer-grade hardware. This preserves transaction atomicity and streamlines execution, allowing the system to coordinate changes cleanly and maintain composability. Solana's success in hosting numerous token launches and generating large on-chain trading volumes reflects the power of this design.

Despite its advantages, Solana's approach now encounters inherent hardware limits. To keep pace with current throughput, validators already require CPUs exceeding 3.1 GHz, more than 500 GB of high-speed RAM, and at least 2.5 TB of high-throughput NVMe storage. With CPU utilization hovering around 30% under heavy load, further gains via software optimization alone are limited. Instead, additional acceleration—such as offloading certain tasks to FPGAs—becomes essential to push performance even higher. Moreover, validators must store and access ever-expanding account data. Standard consumer-grade hardware will inevitably be replaced by specialized solutions, like NVMe-oF, for scaling storage without sacrificing low latency. Network bandwidth similarly faces pressure: peer-to-peer (P2P) communication already consumes around 0.8 Gbps per validator, posing a risk of exceeding the 1 Gbps internet limits typical of consumer-grade networking environments if throughput continues to increase.



Ingress and edge filtering
region A
ingress
edge scrubbing (dedup, validity)
sigveirfy (GPU or FPGA)
region B
region C
tx
low-latency global transit
Pre-execution
replicating cluster 1
replicating cluster 2
replicating cluster 3
executing node 1 data node 1
executing node 2 data node 2
executing node 3 data node 3
executing node 4 data node 4
tx with effects
Tx execution planning and routing with FPGA/SDN
account state cache
semantics-aware scheduling
deduplication
hardware offload
simple path
stale-state
bypass execution/state write
main cluster
state flush
sequencing (concurrency-optimized due to planning) and hashing
broadcast TX WITH tx effect
Global PoPs for broadcast
region A
full node
region B
region C
region D
low-latency global transit
RDMA protocol
infini-band switch

Figure 1: System architecture

To address these emerging bottlenecks, we advocate for a multi-executor, hardware-accelerated SVM network. The design is shown in Figure 1. This design scales a single-state blockchain infinitely by distributing workload across specialized hardware and clusters while preserving a global atomic state. The following sections outline how this architecture decomposes critical pipeline stages into microservices to scale non-atomic components like ingress and state publishing, hardware acceleration for scheduling and locking, a sharded database for scaling storage, and a new hybrid consensus protocol based on Solana that guarantees both security and low latency. To further increase usability, SOLAYER CHAIN introduces chain-level supports of hooks, cross-chain contract calls, jumbo transactions, seamless wallet support, and OAuth-based signers. Overall, SOLAYER CHAIN aims to become the most user-friendly chain with 1M+ TPS and 100Gbps+ bandwidth.

2 Background

In the following subsections, we introduce the background of technology leveraged by SOLAYER CHAIN.

2.1 Solana and SVM

Solana introduces a novel blockchain architecture that fundamentally departs from traditional consensus mechanisms through its proof-of-history (PoH) protocol and parallel execution environment. The SVM employs a shared-nothing architecture where each transaction's read and write sets are determined prior to execution. This enables the runtime to construct a dependency graph that maximizes parallel execution while maintaining serializability guarantees. Unlike the Ethereum Virtual Machine's sequential execution model, SVM processes non-conflicting transactions concurrently across available CPU cores. The SVM's memory model implements a multi-version concurrency control (MVCC) mechanism, maintaining multiple versions of data to support concurrent read operations while ensuring write operations maintain strict serialization order based on the PoH sequence.

The EVM and SVM exemplify contrasting approaches to concurrency control. The EVM employs optimistic concurrency control, where transactions are processed concurrently under the assumption that conflicts will be rare. However, because transactions do not declare their state access upfront, conflicts—such as multiple transactions attempting to modify the same state—are only detected at runtime, requiring rollbacks and retries. In contrast, the SVM adopts pessimistic concurrency control, where transactions must declare their state access patterns upfront. This allows Solana's Sealevel parallel execution engine to execute non-overlapping transactions concurrently, avoiding conflicts altogether. By combining explicit state access declarations with advanced memory and execution models, Solana achieves unparalleled scalability and efficiency in blockchain processing.

2.2 RDMA and Infiniband

Remote Direct Memory Access (RDMA) enables zero-copy data transfer directly between application memory spaces across network-connected systems. Modern RDMA implementations leverage specialized network interface cards (RNICs) that implement both the transport protocol and direct memory operations in hardware. This architecture bypasses traditional operating system networking stacks, eliminating context switches and CPU overhead associated with network I/O processing. The RDMA protocol suite implements both connection-oriented (Reliable Connected) and connectionless (Unreliable Datagram) transport modes, with the former providing guaranteed in-order delivery and the latter optimizing for low-latency scenarios where applications can tolerate potential packet loss. The protocol's memory semantics enable remote memory operations to complete without interrupting the remote CPU, achieving end-to-end latencies below one microsecond for small messages in optimized deployments. This is achieved through a sophisticated protection model where memory regions must be explicitly registered with the RNIC, which then maintains page tables for direct virtual-to-physical address translation.

InfiniBand is a prominent implementation of RDMA, offering a high-speed, low-latency networking architecture tailored for high-performance computing and data center environments. It utilizes Host Channel Adapters (HCAs) to offload protocol processing and memory translation to hardware, thereby reducing CPU involvement in data transfer operations. InfiniBand supports data transfer rates ranging from 10 Gb/s (Single Data Rate) up to 100 Gb/s per port, using both copper and optical fiber connections. This high bandwidth, combined with ultra-low latencies—as low as 600 nanoseconds end-to-end—makes InfiniBand particularly effective for applications requiring rapid data exchange, such as artificial intelligence and high-performance computing workloads. Its efficiency and scalability have led to widespread adoption in distributed machine learning and other data-intensive applications.

2.3 Software-defined Network

Software-defined Networking separates the control plane from the data plane, enabling programmatic control over network behavior through a logically centralized controller. Modern SDN architectures implement a three-tier model: the data plane consists of programmable forwarding elements that process packets based on match-action rules, the control plane implements network policies and path computation, and the management plane provides high-level network orchestration. The forwarding plane has evolved from fixed-function match-action tables to fully programmable packet processing pipelines through languages like P4, enabling custom protocol implementation and complex packet transformations at line rate. SDN controllers implement distributed consensus protocols to maintain network state consistency across controller replicas, while providing northbound APIs that abstract network complexity for applications. The forwarding plane typically employs a pipeline architecture where packets traverse multiple match-action stages, with each stage capable of performing arbitrary modifications to packet headers and maintaining local state. This programmability enables implementation of complex network functions like load balancing, traffic engineering, and network virtualization directly in the data plane while maintaining line-rate performance.

3 Scaling Transaction Processing

Overall architecture of SOLAYER CHAIN is illustrated in Figure 1. Each transaction reaches an initial ingress point that conducts sigverify and local deduplification. The verified transaction is then sent to the pre-execution cluster that conducts pre-execution. We discuss this pipeline in subsection 3.1. Transaction effect and intermediate snapshots are sent to the sequencer via Infiniband. The sequencer leverages SDN switch and additional FPGA to decide whether a transaction shall go through simple path (i.e., all transaction accounts are at latest version when pre-executing) or complex path (i.e., at least one account has a newer version). For simple path transaction, the state change is directly applied via RDMA, with a local cache on the SDN. For complex path transaction, it enters a local mempool with sub-millisecond processing speed. The sequencer schedules the transactions in the local mempool to achieve fair and optimal parallel execution of all transactions. The scheduling algorithm is described in subsection 3.2. We discuss distributed database that stores account data in subsection 3.3. After the transaction is executed and state change has been written, the transaction is propagated through a global PoPs.

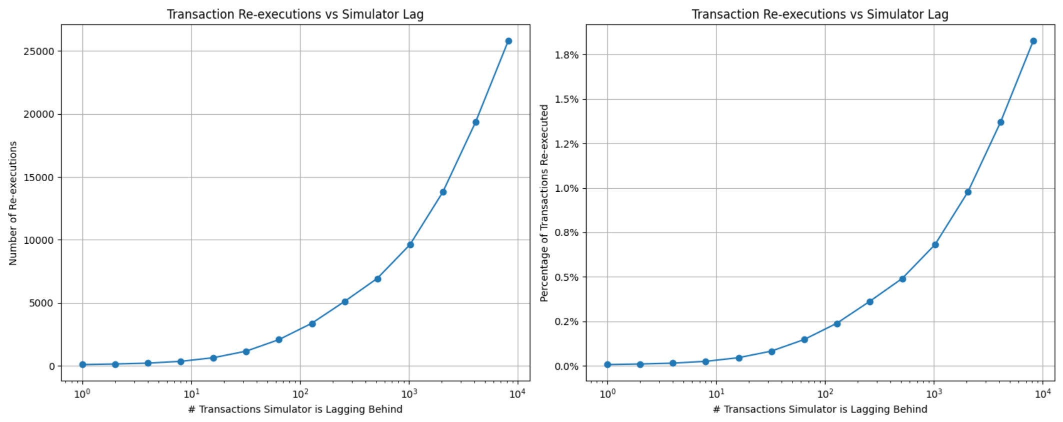


Figure 2: Amount of transaction requiring re-execution versus the amount of transactions the executor is lagging behind.

3.1 Microservice Pipeline

SOLAYER CHAIN's transaction processing pipeline mirrors some of Solana's core stages: signature verification (sigverify), deduplication, scheduling, banking, and storage. Banking stages are inherently sequential, requiring either coarse-grained locking or single-threaded execution to maintain consistency. However, while Solana executes these stages in a monolithic architecture, signature verification, deduplication, and storage operations can be decoupled and distributed. SOLAYER CHAIN exploits this observation by decomposing these stages into independent microservices deployed across an elastic compute fabric. Our system implements dynamic resource provisioning through a feedback-driven control plane that monitors transaction ingress rates and automatically scales processing capacity.

A key insight in our design is that a significant portion of transactions can be pre-executed independently, provided they do not exhibit read-write dependencies on accounts accessed by concurrent transactions (i.e., no dirty read happens). According to Figure 2, a simulation of pre-executing transactions from 1000 slots starting from 304992000, only 2% of the transaction needs to be re-executed due to dirty read, even if the executor is significantly lagging behind the chain.

Building on this observation, SOLAYER CHAIN introduces a simulation stage that precedes transaction scheduling. This stage, which can be horizontally scaled across nodes, performs speculative execution of transactions against the most recently committed state on the node. During simulation, the system captures both the transaction effects and a set of intermediate execution snapshots at account access boundaries. While some transactions may exhibit conflicts due to overlapping account access patterns, our approach allows parallel pre-execution of the non-conflicting majority.

Notably, transactions that only perform read operations can be fully validated and confirmed at the edge, bypassing the central banking stage entirely. To further reduce the congestion of conflicting transactions, transactions accessing hot accounts can be pre-executed for all possibilities of the future value of those hot accounts. Each account has a short-term prediction model implemented using Winter-Holt's double exponential smoothing prediction (DESP) for each byte in the account. Depending on the access frequency of an account, the pre-execution node operator could manually inject a more accurate prediction model, and a transaction can be simulated millions of times with different possible account data. For the remaining transactions that are still conflicting with previous measures, when conflicts are detected, the subscriber of the simulation results can rapidly reconstruct the correct execution state from the nearest valid snapshot in simulation results, eliminating the need for complete transaction re-execution. This design significantly reduces the computational overhead traditionally associated with the banking stage while maintaining strict consistency guarantees.

3.2 Transaction Semantic Aware Scheduling

Solana's transaction processing model employs account-level access pattern analysis to batch transactions, preventing dirty reads through strict isolation. The scheduler partitions transactions into batches based on their declared account access patterns, optimizing for parallel execution. While this approach enables lock-free execution within batches during the banking stage, it is inherently conservative, treating all account accesses within a transaction as concurrent. SOLAYER CHAIN extends this model by introducing fine-grained sequence prediction of read-write operations. By analyzing the temporal ordering of account accesses within transactions, our system constructs an optimized locking schedule that permits concurrent execution of transactions accessing the same accounts when their actual read-write sequences do not conflict during execution. This dynamic scheduling approach significantly reduces lock contention while maintaining serializability guarantees.

SOLAYER CHAIN leverages its simulation stage to obtain estimated read-write sequences of transactions before they enter the scheduling phase. Formally, for a transaction t, we define its execution trace as a sequence:

$$E(t) = \{(op_1, a_1, c_1), (op_2, a_2, c_2), ..., (op_n, a_n, c_n)\}$$

where $op_i \in \{read, write\}$ represents the operation type, a_i represents the accessed account, and c_i represents the predicted execution time cost between op_i and op_{i+1}. For any two transactions t_i and t_j with overlapping accounts $A(t_i) \cap A(t_j)$, we define the temporal sequence $S_a(t)$ for account a as the ordered sequence of operations $\{(op_k, t_k) | a_k = a\}$ where t_k is the relative timestamp. The computational gap between consecutive operations is defined as $G_k = c_k \cdot \alpha$ where α is the average instruction execution time. Two transactions can execute in parallel if for all $a \in A(t_i) \cap A(t_j)$, their sequences $S_a(t_i)$ and $S_a(t_j)$ have no write-write conflicts, and for any read-write pairs $(read, t_k) \in S_a(t_i), (write, t_l) \in S_a(t_j)$, the gap satisfies $G_k > \delta$ where δ is a tunable number representing the estimation error that simulations can make. Note that while the goal is to find a lock free schedule, lock is still used during execution as the schedule is based on estimation and may not be correct.

Given that this problem is analogous to bin-packing problem, the best algorithm to solve the optimal schedule is NP-hard. To ensure the schedule can be found at sub-millisecond level, we leverage Shortest Makespan First (SMF), a greedy algorithm that is widely used in database systems, for solving a near-optimal schedule.

In addition, SOLAYER CHAIN employs a parallel scheduler ensemble that concurrently explores multiple scheduling strategies to optimize transaction throughput. The ensemble includes Solana's multiple original account-based partitioning algorithms as a baseline, along with the SMF algorithm. Each algorithm in the ensemble generates a candidate schedule with an estimated time cost, and the scheduler selects the schedule with the least time cost.

3.3 Sharding Database with RDMA

Blockchain systems face significant scalability challenges due to their ever-growing state size. For example, Ethereum's state currently exceeds 1.1 TB, highlighting the need for efficient state management solutions. While storing this data in memory is crucial for low-latency access, the memory limitations of individual servers necessitate a distributed approach. SOLAYER CHAIN implements a sophisticated sharding mechanism based on a key-value store architecture, mapping 32-byte account addresses to their corresponding state data. Unlike Solana, which imposes a 10MB limit on account data size, SOLAYER CHAIN supports arbitrary data sizes per account.

Each database node in SOLAYER CHAIN stores a shard of data. The core architecture of each database node of SOLAYER CHAIN consists of three primary components. First, a memory-resident jump table maintains mappings between account addresses and their corresponding memory addresses, data length, and version number (i.e., how many times the account has been written). Second, a contiguous data region stores the actual account state data along with associated synchronization primitives and metadata. We observe that most state change on Solana do not modify the length of data in the account. Under rare occurence of state size change, the account data may not be stored in contiguous memory, and later gets redistributed to the contiguous memory during routine rebalancing. Third, an in-memory local cache maintains frequently accessed account data using an LRU (Least Recently Used) eviction policy to reduce network round-trips for popular accounts.

To enable efficient cross-node data access by the executor, SOLAYER CHAIN leverages InfiniBand RDMA (Remote Direct Memory Access) protocol. This approach offers several advantages: ultra-low latency (ns-level) data access across nodes, bypass of operating system overhead, reduced CPU utilization during data transfer, and zero-copy data movement between nodes. The RDMA infrastructure allows SOLAYER CHAIN to maintain high performance even when data needs to be fetched from remote nodes, ensuring consistent transaction processing speeds across the distributed system.

The system implements dynamic load balancing through a background rebalancing mechanism that operates during periods of low network activity. This process monitors account access patterns, analyzes historical memory access footprints, and redistributes data across nodes to optimize locality and minimize cross-node data fetches for common transaction patterns. The rebalancing process considers multiple factors, including access frequency of accounts, data size, network topology, node capacity constraints, and current load distribution. This adaptive approach ensures that frequently co-accessed data resides on the same node, reducing the need for remote data access and improving overall transaction throughput.

4 Scaling Consensus

Existing rollup-based designs typically offload verification to a large set of commodity validators that must reconstruct or dispute transactions posted on Layer 1 (L1). However, verifying transaction streams at 1Gb/s is beyond the capacity of most commodity nodes. Moreover, posting 1Gb worth of data on any data availability (DA) layer or L1 is prohibitively expensive, saturating available bandwidth and raising on-chain fees. Consequently, we introduce a Proof-of-Authority-and-Stake (not to be confused with Proof of Stake and Authority or PoSA) architecture: a trusted entity acts as a sequencer (leader) and publishes shreds (a set of transactions in the block), and every prover stakes and votes to decide whether a shred can be accepted. Solana is used as a fallback consensus venue when the sequencer is behaving maliciously.

Our protocol batches transactions into shreds, each containing a slot number, a vector of transactions, version metadata for accessed accounts, and linkage hashes (e.g., last shred version). The presence of account versions is critical because it allows each shred to indicate the exact state context in which the sequencer executed the transactions. By referencing these state versions, any node that receives the shred can re-construct the relevant portion of the ledger and compute the resulting "effect hash"—the cryptographic digest of post-transaction states. Only a minimal (Effect Hash, Shred Hash) pair is published on Solana to ensure data availability, while most of the bulk data is propagated to each prover. This design caps on-chain overhead and avoids saturating L1 capacity.

Upon receiving a shred, a prover checks if its local ledger contains the correct account version. If not, it requests the missing shreds necessary to build up the needed state from the leader. Then, the prover re-executes all transactions in the shred, deriving the effect hash. If the locally computed hash matches the shred's embedded effect hash, the prover votes for acceptance on SOLAYER CHAIN. Once a shred accumulates votes from 51% of the selected provers, the sequencer assembles a proof for the shred and marks the shred as finalized if all previous shreds are finalized.

Should the sequencer propose invalid or malicious shreds, honest provers will detect discrepancies between the expected and computed effect hashes and vote against them. It is impossible to include invalid transactions in the sequencer itself. Repeated offenses mark the sequencer as offline, triggering a failover to a backup sequencer from a proof-of-authority set. The re-election occurs on Solana as a down sequencer, which means SOLAYER CHAIN can no longer process transactions. All honest provers vote for the re-election and advance to the next sequencer once the vote reaches 2/3. If 2/3 of the provers are not malicious, re-election can finish in seconds. Furthermore, to curb dishonest or lazy provers, the sequencer periodically broadcasts intentionally invalid shreds: any prover that "blindly" votes for such a shred is automatically slashed and excluded from further rounds. This mechanism deters passive acceptance of shreds and ensures each prover consistently re-executes transactions. To combat censorship resistance, if the sequencer repeatedly ignores a transaction (e.g., censorship), any user can insert the ignored transaction into a future shred by sending the transaction data on Solana to the relevant program; the sequencer's refusal to accept a transaction on Solana similarly flags it as unresponsive or malicious.

As it is unrealistic for provers to use costly HPC infrastructure and hardware accelerators to follow the chain and verify every shred, the sequencer leverages the round-robin method to randomly select only 2/3 of the online prover set to execute the shred. The provers further subdivide the task to different nodes they possess in a round-robin method. If a prover has 10 nodes, each node only needs to handle 1/15 of all shreds. Provers can use the cloud to scale the nodes to handle more traffic elastically. Note that to achieve a 51% vote, only 4/5 of the provers selected have to vote for the shred, allowing downtime of provers. Under edge cases, when more than 1/5 of the provers selected are down, the sequencer should select more provers to join the verification of the shred. Failure to do so would make the sequencer marked as down. The sequencer has no incentive to conduct denial of service (e.g., always selecting the same provers to verify all shreds) for the provers because if the sequencer can no longer finalize blocks, it is marked as down.

Provers receive fees from processed shreds and inflationary $LAYER rewards as incentives to participate. However, if they exhibit malicious behavior or repeatedly fail to process shreds, they face tiered slashing penalties and get evicted from the prover set until manual re-joining. The first violation results in the loss of that epoch's accrued fees. A second violation in that epoch incurs a 1% slash on staked tokens, and each subsequent violation within that epoch leads to a 5% stake slash.

5 Improving User Experience

5.1 Wallet-agnostic and DApp-first integration

Traditional chain integrations require support at both the wallet and dApp layers. For EVM L2s, this is because EIP-155 mandates that the chain ID be included in the signature payload, which wallets enforce. However, SVM transaction signatures do not include chain IDs within the transaction structure. Instead, each transaction contains a recent block hash, and replaying the transaction on other chains simply results in its rejection. Secondly, in EVM ecosystems, wallets are responsible for transaction broadcasting, whereas in SVM, the dApp is responsible for broadcasting transactions.

Using SOLAYER CHAIN 's SDK, dApps can allow users to create transactions directly on SOLAYER CHAIN using any Solana-compatible wallet without requiring explicit wallet support. While direct wallet support remains desirable for assets to be displayed seamlessly, it is not a strict requirement for transaction execution and user onboarding.

5.2 Hooks

In many on-chain environments, re-executing transactions to detect post-execution opportunities — such as arbitrage, liquidations, or real-time account indexing—can be computationally expensive for off-chain entities. To address this, we introduce Hooks: a mechanism that automatically executes user-defined logic immediately after a transaction interacts with one or more on-chain programs. Conceptually, Hooks serve as a built-in "backrunning" layer: once a targeted program has updated its state, one or more Hooks can trigger arbitrage checks, liquidation actions, accounting operations, or any other use case requiring timely state awareness.

Technically, a Hook has the same definition as a transaction: Hook users provide accounts and data, and gas is charged from the designated account each time the hook is triggered. The hook can be disabled after three occurrences of lack of funds to pay gas in the fee payer account. SOLAYER CHAIN provides a precompile that manages Hook registration and bidding. Users submit bids to attach their Hooks to specific programs, competing in a Dutch-auction-like model where the top 16 bids (per program) are eligible for execution during the following epoch. If a transaction touches multiple programs with registered Hooks, only 16 total Hooks (sorted by descending bid) will run. Once the system completes the transaction, these Hooks are called in sequence, and the bid for each Hook is split: 40% to the transaction initiator, 40% to the owner of the program, and 20% to the network.

This incentive model reduces the cost of exhaustive replay while encouraging ecosystem participants to incorporate Hooks. The 40-40-20 distribution ensures that both end-users and program owners benefit from higher bids while the network collects a portion to offset additional on-chain overhead. By embedding Hooks directly into the execution pipeline, SOLAYER CHAIN also mitigates spam or off-chain MEV exploitation, as real-time logic becomes accessible to any party willing to bid. Overall, Hooks provides a fair and performance-oriented approach to automating post-transaction actions without overburdening the core network.

5.3 Cross-chain Contract Calls

SOLAYER CHAIN has a bi-directional native bridge implemented inside the sequencer to relay messages and assets from Solana to SOLAYER CHAIN. The bridge ignores Solana re-org by implementing an insurance fund that automatically covers loss due to re-org. If the total value of the asset (measured through an oracle) relayed before the finalization is lower than the insurance fund size, the asset becomes immediately available on SOLAYER CHAIN. Otherwise, the asset becomes available after the transaction is finalized. All messages can be relayed instantly (ignoring re-org) or after finalization by setting the flag in the transaction. The receiver hook is automatically invoked once the message becomes available and can call additional programs.

SOLAYER CHAIN also introduces a cross-chain program call bridge through account mirroring, where each SOLAYER CHAIN program and account has a corresponding Program Derived Address (PDA) on the Solana mainnet. Central to this design is the MainnetCall instruction type, enabling atomic cross-chain operations orchestrated via a built-in system program. Whenever a SOLAYER CHAIN program invokes MainnetCall, the transaction routes instructions to its associated PDA on Solana. In doing so, programs can transfer funds, dispatch cross-chain function calls, and even execute complex on-chain logic spanning both networks, all within a single atomic operation. To ensure atomicity, MainnetCall instruction never reverts (even when the Solana transaction is reverted) and can be inserted only at the end of the transaction but not in between so that when

Solana re-org occurs, there is no impact, as the Solana transaction only needs to be broadcasted again until it is finalized. By combining both bridge, one can implement solutions including but not limited to single transaction cross-chain swap and cross-chain yield vault rebalancing.

5.4 Jumbo Transaction

To support complex transaction logic, especially for hook transactions, SOLAYER CHAIN introduces jumbo transactions. This new transaction type significantly increases the transaction size limit, allowing for more cross-program invocations. Fees for jumbo transactions grow exponentially as the transaction size increases. By leveraging jumbo transactions, users can read and write thousands of accounts, execute thousands of instructions, or deploy multiple programs within a single transaction.

5.5 ZK-Login

Users can use their Google, X, Reddit, or any service that support OAuth as a wallet. SOLAYER CHAIN has native support for transactions signed using OAuth. The underlying workflow is similar to Sui's zkLogin.