# COSINE: A Dynamic Layer-2 Protocol for Real-Time Trustless Credit Scoring and Fraud Mitigation

Dimoris Chinyui

February 2, 2025

## Table of Contents

### Contents

## 1. Abstract

The decentralized finance (DeFi) ecosystem has experienced significant security challenges since its inception, with documented losses exceeding US$100 billion due to various forms of financial fraud, including sophisticated social engineering attacks, protocol exploitation, and malicious contract deployment. Additionally, approximately US$150 million has been lost through defaults in under-collateralized cryptocurrency lending protocols. These security vulnerabilities, combined with the irreversible nature of blockchain transactions, have created persistent barriers to mainstream adoption while contributing to market instability. To address these challenges, we present COSINE, a novel Layer-2 protocol that implements dynamic, cross-chain and trust-minimized credit scoring mechanisms. This protocol provides real-time creditworthiness assessment across all blockchain networks, offering an efficient and transparent evaluation framework that maintains decentralization principles.

At the core of COSINE's innovation is a **sequential voting mechanism** that incorporates **reputation-weighted votes**, allowing users to influence credit scores in real-time based on their historical voting behavior. This ensures that accuracy in determining trustworthiness scales as the network grows, adapting and refining its assessment capabilities with each new transaction. The protocol also integrates advanced **fraud detection** techniques, such as **randomized time windows** and **multi-hop association risk analysis**, to identify malicious actors and penalize wallets associated with suspicious behavior.

Instead of using traditional numeric credit scores, COSINE employs **cosine similarity** in a vector space, offering a more stable and intuitive measure of trustworthiness and risks when transacting.

In addition, COSINE enables cross-chain wallet linking, allowing credit assessments to reflect activity across multiple blockchains, thereby improving interoperability and trust across the DeFi ecosystem. The protocol's self-tuning nature ensures that credit scoring parameters adapt to evolving network conditions, while its cost-recoup mechanism guarantees that users who request credit score verifications contribute to the ongoing sustainability of the system.

COSINE ultimately provides a transparent, flexible, and adaptive credit scoring system that enhances DeFi's real-world adoption and trustworthiness, allowing users transacting all digital currencies feel secure.

## 2. Motivation & Key Goals

The rapid growth of decentralized finance (DeFi) has created immense opportunities, but also significant challenges, particularly in areas of **fraud prevention**. Traditional credit scoring systems are deeply centralized, biased, and often exclude individuals from financial access. In contrast, DeFi relies on **smart contracts** and **open-source protocols**, which are transparent and accessible to all, but lack efficient ways to assess the trustworthiness of participants and prevent malicious activities. To address these problems, COSINE was designed with the following key goals:

- **Real-Time On-Chain Credit Scoring & Verification:** One of the biggest hurdles in DeFi is the difficulty of facilitating **under-collateralized lending**, where loans are issued without the need for excessive collateral. COSINE provides a real-time, **trust-minimized credit score** for every wallet, allowing decentralized applications (dApps) and lenders to assess the creditworthiness of users instantly. This **reputation-based scoring** helps foster confidence in lending systems, reducing reliance on traditional credit bureaus and expanding financial inclusion.

- **Dynamic, Self-Tuning System:** In a fast-moving DeFi environment, traditional **fixed penalty systems** are inadequate. COSINE uses a **self-tuning mechanism** that adjusts penalty and reward factors dynamically through **Kalman filters**, learning from real-time data with enhanced responsiveness. Kalman filters provide adaptability to sudden network changes and handle noisy data more effectively, ensuring that credit scoring parameters evolve efficiently with the network. This continuous adaptation allows the protocol to respond to new fraud tactics with greater agility.

- **Reputation-Weighted Voting for Community-Driven Governance:** Unlike traditional centralized systems, COSINE leverages **community-driven governance** through **reputation-weighted voting**. Each participant's ability to influence the protocol's decisions (e.g., vote weight) is based on their **track record** of accurate and honest votes. This incentivizes **consistent, truthful contributions**, aligning participants' interests with the integrity of the ecosystem. Over time, this creates a **robust, decentralized** mechanism for credit scoring and fraud detection.

- **Advanced Fraud Detection:** Fraudulent activities, such as **money laundering** or **scams**, are rampant in the blockchain industry. COSINE addresses this by implementing **randomized time windows** and **multi-hop analysis**, which makes it more difficult for attackers to predict and evade detection. By penalizing wallets associated with malicious behaviors or addresses, COSINE creates a system that **disincentivizes bad actors** and helps maintain the integrity of the network.

- **Cosine Similarity for Trust Assessment:** Traditional credit scores are often **volatile** and prone to sudden fluctuations, leading to unfair judgments. COSINE introduces **cosine similarity** to measure how close a wallet's credit score is to a desired threshold or to another wallet's score, providing a more **stable and intuitive** measure of trustworthiness. This method reduces score volatility and offers clearer insights into a wallet's risk profile, aiding both users,lenders, and borrowers in making better decisions.

- **Cross-Chain Interoperability:** DeFi operates across a range of **blockchains**, but credit scoring and fraud detection tools often fail to integrate across chains. COSINE solves this problem by allowing wallets to link across multiple chains, enabling **cross-chain trust metrics** that reflect a user's behavior across different ecosystems. This creates a unified, **global** credit score, enhancing the flexibility and **interoperability** of the DeFi space.

By combining **real-time scoring**, **adaptive learning**, **community governance**, and **advanced fraud mitigation**, COSINE addresses the pressing need for a **trust layer in DeFi**. The system solves major pain points, such as the lack of **adaptability**, **transparent credit assessments**, and effective **fraud detection**, all while reducing dependency on centralized institutions. Ultimately, COSINE paves the way for a more **trustworthy, inclusive, and secure DeFi ecosystem**.

## 3. Protocol Overview

This section provides a broad technical overview of the COSINE protocol, focusing on the transition to a **sequentially updated**, **self-tuning** design that processes voting events in real time and adjusts scaling factors automatically via continuous learning.

### 3.1. Core Principles & Key Components

1. **Credit Score ($C_W$):** Each wallet $W$ on the L2 has a scalar credit score $C_W$. The protocol updates this score incrementally over time based on:
   - *Sequential Vote Events*: Negative or positive votes cast by other users, each weighted by the voter's reputation $R_U$.
   - *Association Risk Penalties*: Based on random time windows/hop limits identifying suspicious links to malicious or blacklisted wallets.
   - *Rehabilitation*: Positive votes that push a wallet's score back up if it demonstrates better behavior or was wrongfully penalized.
2. **Voter Reputation ($R_U \in [0, 1]$):** Each user $U$ who participates in governance or malicious-flagging votes has a reputation metric. When the user's voting aligns with final consensus outcomes, $R_U$ increases; when they oppose consensus outcomes, it decreases. Over time, reliable and consistent voters gain more influence.
3. **Self-Tuning Scaling Factors:** Rather than relying on static thresholds (e.g., "Major penalty" or "Minor penalty"), COSINE employs dynamic scaling factors $\{K_{\text{neg}}, K_{\text{pos}}, K_{\text{assoc}}, K_{\text{rehab}}\}$. These factors are updated in real time using a Kalman filter-based feedback loop that dynamically adjusts each scaling factor based on the difference between observed and expected impacts of credit score shifts. This approach enables the protocol to respond rapidly to changing network conditions and handle noisy data more effectively than traditional methods, ensuring that score adjustments are appropriately scaled according to current network behavior. The Kalman filter's recursive nature also maintains computational efficiency suitable for real-time processing.
4. **Random Time Windows & Hop Limits:** For association risk analysis, the protocol chooses random intervals and a random hop depth in an attempt to thwart malicious actors who try to carefully plan or space out their illicit transactions. If suspicious transactions appear in that random window or within that hop limit, partial penalties are assigned.
5. **Cosine Similarity Verification:** When a lender or any other user wants to check a wallet's creditworthiness, the system computes a *two-dimensional embedding* of the wallet's normalized credit score and compares it (via cosine similarity) to a chosen threshold vector. A similarity near 1 indicates the wallet is sufficiently trustworthy relative to that threshold.
6. **Sequential Processing & Low Latency:** Each new vote event or suspicious association triggers immediate, low-latency updates on the L2 chain, avoiding heavy cryptographic operations.

### 3.2. Data Sources & Measured Variables

The protocol relies on direct blockchain data to measure:

- **Transaction Amounts** ($A_i$), **Timestamp** ($t_i$), and **Clustering** ($n_{\textbf{cluster}}$) from normal on-chain activity.
- **Vote Events**, each storing the voter's address, the sign of the vote ($-1$ or $+1$), and the block timestamp of the vote.
- **Association Links** to detect multi-hop or direct transactions with known malicious wallets.

From these measurements, the protocol derives rolling means, standard deviations, and other variables ($\mu_V(t)$, $\sigma_V(t)$, $\mu_R(t)$, etc.) using exponential moving averages.

### 3.3. Sequential Updates

COSINE does not wait for large batch voting windows. Instead, *every* vote and association risk event triggers a credit score update in near-real time. This is achieved by:

- Maintaining rolling EMAs of historical vote deviations, credit score shifts, and so on.
- Updating these EMAs on every event, ensuring the system always "learns" from the most recent data.

This help the protocol maintain responsiveness and adaptability in a dynamic network environment, which is crucial for scalability

### 3.4. Validator Roles & VRF-Based Subset Selection

Although the primary emphasis in this whitepaper is on the self-tuning credit scoring mechanism, COSINE still inherits a *VRF-based* validator selection process (see Section 4). A subset of validators is chosen at random (weighted by stake and performance scores), each sequential update is processed, and a consensus update is then posted to the L2 ledger. This ensures trust-minimized, decentralized operation.

## 3.5. Storage & Dynamic Access

All credit scores $C_W$ reside with every validator node on the COSINE L2 for maximum transparency. However, direct reading of the raw score is avoided:

- For *official verifications*, the user or dApp calls a function to compute the *cosine similarity* between the wallet's normalized score vector and a threshold vector (Section 10).
- The fee mechanism ensures repeated updates do not go unpaid (Section 12).

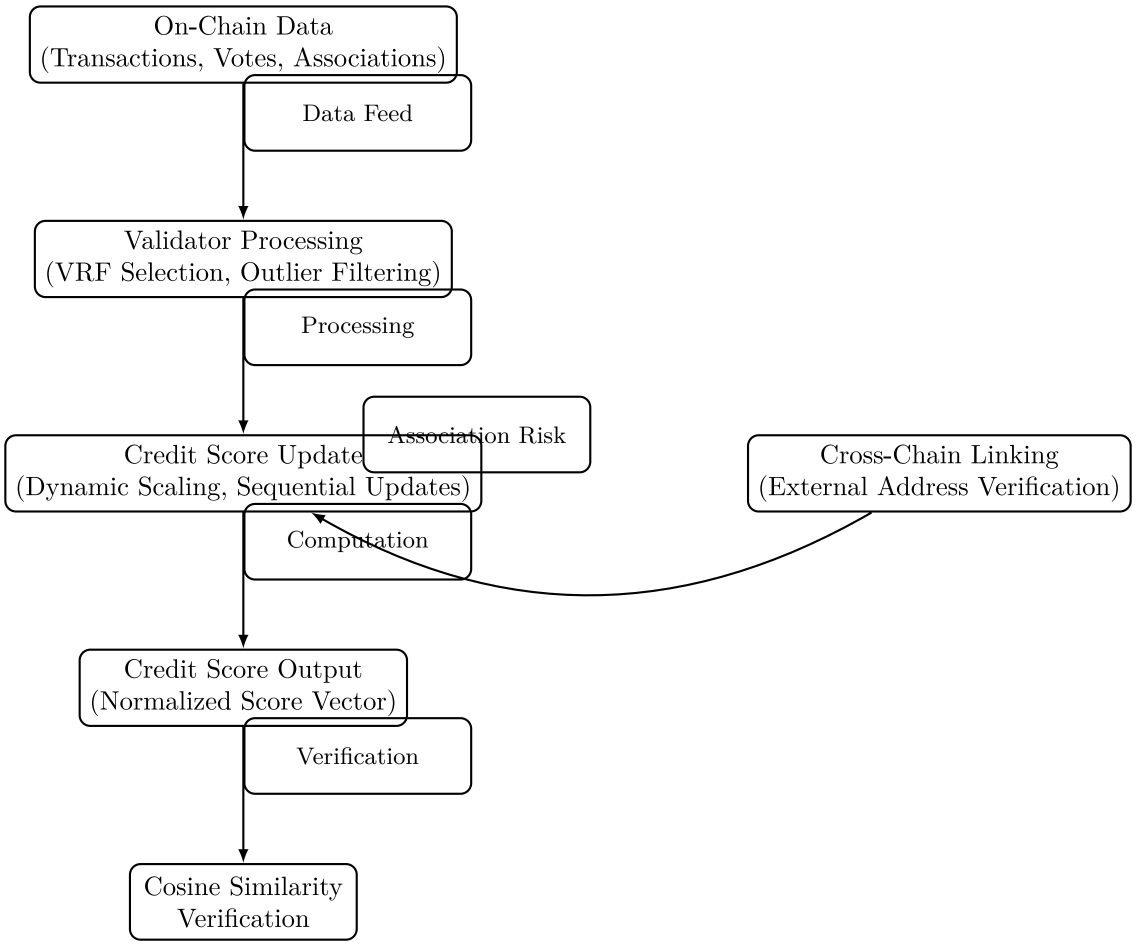

Figure 1: COSINE Protocol Overview

The subsequent sections detail how negative and positive vote events shift the score, how association risk triggers partial penalties, how community governance operates via sequential voting, how the protocol processes cross-chain data, and how actual workflows (borrowing, verifying, re-linking) unfold end-to-end.

## 4. Mechanics of Validator Operations

While the new approach emphasizes dynamic scoring parameters and sequential updates, the core validator architecture for producing the updated credit score remains grounded in VRF-based subset selection and outlier filtering. This section details how these mechanics function, ensuring that final credit scores are agreed upon fairly and with minimal overhead.

### 4.1. Phase 1: Determining Online Validators

First, validators who have staked Cosine tokens must "ping" the system regularly to be deemed online. Let:

$$V_{\text{online}} = \left\{ v \;\middle|\; \text{LastPing}(v) \geq (T_{\text{current}} - \Delta T_{\text{ping}}) \right\},$$

where $\Delta T_{\text{ping}}$ might be 5 minutes. Offline or non-responsive validators are excluded from the selection set.

### 4.2. Phase 2: VRF-Based Subset Selection

From the online validators, the protocol wants to select a fraction $\gamma$ for each credit score update. Define:

$$N_{\text{selected}} = \lceil \gamma \cdot |V_{\text{online}}| \rceil ,$$

bounded by some maximum $N_{\max}$. Each validator $v$ computes

$$y_v = \text{VRF}_{sk_v}(x),$$

where $x$ is a random seed (often a recent block hash). The *reputation-based weight* is

$$W_v = S_v^{\alpha} \times (1 + \beta\, P_v),$$

with:

- $S_v$ = stake,
- $P_v$ = performance score in $[0, 1]$,
- $\alpha \in [0.5, 1]$ and $\beta > 0$ are protocol parameters.

Then define

$$\text{score}_v = \frac{y_v}{W_v}.$$

The subset $V_{\text{selected}}$ is chosen as the top $N_{\text{selected}}$ validators with the smallest $\text{score}_v$.

### 4.3. Phase 3: Credit Score Computation & Outlier Filtering

When an event triggers a wallet $W$'s score update—such as a vote, association penalty, or rehabilitation (see Sections 7, 6, 8)—each validator $v$ in the randomly selected subset $V_{\text{selected}}$ (determined via VRF as detailed in Section 4.2) computes a proposed new credit score $S_v$. This section outlines the aggregation process, using a hybrid mean-median approach with dynamically adjusted thresholds to filter out problematic (e.g., malicious or erroneous) validator proposals and compute a final consensus score $S_{\text{agg}}$.

#### 4.3.1. Step 1: Proposal Submission

Each validator $v \in V_{\text{selected}}$ calculates:

$$S_v = C_W^{\text{old}} + \Delta_v,$$

where:

- $C_W^{\text{old}}$ is the wallet $W$'s previous credit score, stored on the COSINE Layer-2 ledger (Section 3).
- $\Delta_v$ is the proposed shift, derived from the dynamic scaling formulas (Section 8), incorporating vote impacts ($\Delta_{\text{vote}}$), association penalties ($\Delta_{\text{assoc}}$), or rehabilitation adjustments ($\Delta_{\text{rehab}}$). Each validator computes $\Delta_v$ independently based on on-chain data and its own assessment.

The set $\{S_v\}_{v \in V_{\text{selected}}}$ represents all proposals, with $n = |V_{\text{selected}}|$ being the number of validators in the subset, bounded by $N_{\max}$ (Section 4.2).

#### 4.3.2. Step 2: Hybrid Statistical Measures

To robustly aggregate proposals, the protocol computes two sets of statistical measures: mean-based (using $\mu$ and $\sigma$) and median-based (using $M$ and $\text{MAD}_n$).

**Mean-Based Measures**

- **Mean Computation:** Calculate the arithmetic mean of all proposals:
$$\mu = \frac{1}{n} \sum_{v \in V_{\text{selected}}} S_v.$$
This is a simple average, reflecting the central tendency if proposals were normally distributed.
- **Standard Deviation Computation:** Calculate the sample standard deviation:
$$\sigma = \sqrt{\frac{1}{n} \sum_{v \in V_{\text{selected}}} (S_v - \mu)^2}.$$
Here, $(S_v - \mu)^2$ measures each proposal's squared deviation from the mean, and the square root normalizes it to the same units as $S_v$, providing a scale of variability.

### Median-Based Measures

- **Median Computation:** Sort $\{S_v\}$ in ascending order to obtain $S_{(1)}, S_{(2)}, \dots, S_{(n)}$, then compute:

$$M = \text{median}(\{S_v\}) = \begin{cases} S_{((n+1)/2)}, & \text{if } n \text{ is odd,} \\ \frac{S_{(n/2)} + S_{(n/2+1)}}{2}, & \text{if } n \text{ is even.} \end{cases}$$

  The median is the middle value (or average of two middle values), offering a robust central tendency less affected by extreme proposals.

- **MAD Computation:** Calculate the Median Absolute Deviation:

  1. Compute absolute deviations from the median: $d_v = |S_v - M|$ for each $v$.
  2. Sort $\{d_v\}$ to get $d_{(1)}, d_{(2)}, \dots, d_{(n)}$.
  3. Compute the median of deviations:

$$\text{MAD} = \text{median}(\{d_v\}) = \begin{cases} d_{((n+1)/2)}, & \text{if } n \text{ is odd,} \\ \frac{d_{(n/2)} + d_{(n/2+1)}}{2}, & \text{if } n \text{ is even.} \end{cases}$$

  4. Normalize with a consistency factor for a normal distribution: $\text{MAD}_n = 1.4826 \cdot \text{MAD}$, where 1.4826 ensures $\text{MAD}_n$ approximates the standard deviation under normality.

### 4.3.3. Step 3: Dynamic Threshold Estimation with Kalman Filters

To adapt to network conditions, thresholds $\tau(t)$ (for mean-based filtering) and $k(t)$ (for median-based filtering) are updated dynamically using Kalman filters, treating them as state variables.

### State and Measurement Models

- **State Evolution:** Model thresholds with a random walk:

$$\tau(t) = \tau(t-1) + w_\tau(t), \quad w_\tau(t) \sim N(0, Q_\tau),$$

$$k(t) = k(t-1) + w_k(t), \quad w_k(t) \sim N(0, Q_k),$$

  where $Q_\tau$ and $Q_k$ are process noise variances, reflecting expected threshold variation per update.

- **Measurements:** Define observed threshold requirements based on proposal spread:

$$z_\tau(t) = \frac{\max_{v \in V_{\text{selected}}} |S_v - \mu|}{\sigma},$$

$$z_k(t) = \frac{\max_{v \in V_{\text{selected}}} |S_v - M|}{\text{MAD}_n}.$$

  These represent the largest normalized deviations, indicating how wide the thresholds should be to encompass honest proposals.

**Kalman Filter Steps** For each update event $t$ (triggered by a score update):

1. **Prediction:**
$$\tau^-(t) = \tau(t-1), \quad P_\tau^-(t) = P_\tau(t-1) + Q_\tau,$$
$$k^-(t) = k(t-1), \quad P_k^-(t) = P_k(t-1) + Q_k,$$
where $P_\tau$ and $P_k$ are error variances for $\tau$ and $k$.

2. **Kalman Gain:**
$$G_\tau(t) = \frac{P_\tau^-(t)}{P_\tau^-(t) + R_\tau}, \quad G_k(t) = \frac{P_k^-(t)}{P_k^-(t) + R_k},$$
where $R_\tau$ and $R_k$ are measurement noise variances.

3. **Update:**
$$\tau(t) = \tau^-(t) + G_\tau(t) \cdot (z_\tau(t) - \tau^-(t)),$$
$$k(t) = k^-(t) + G_k(t) \cdot (z_k(t) - k^-(t)).$$

4. **Error Variance Update:**
$$P_\tau(t) = (1 - G_\tau(t)) \cdot P_\tau^-(t), \quad P_k(t) = (1 - G_k(t)) \cdot P_k^-(t).$$

**Initial Default Values and Rationale**

- **$\tau(0) = 2.5$, $k(0) = 3$:** Initial thresholds are set to statistical norms for anomaly detection under a normal distribution. $\tau(0) = 2.5$ covers 98.76% of data (two-tailed), balancing sensitivity and specificity as justified previously (original Section 4.3). $k(0) = 3$ covers 99.7% for MAD, providing a slightly stricter initial filter due to MAD's robustness, ensuring conservative outlier rejection at startup.
- **$P_\tau(0) = P_k(0) = 1$:** Initial error variances reflect moderate uncertainty in $\tau$ and $k$ before data refines them. A value of 1 (unit variance) is a common starting point, allowing rapid adaptation without assuming excessive precision.
- **$Q_\tau = Q_k = 0.01$:** Process noise variances assume small threshold changes per update (e.g., $\sqrt{0.01} = 0.1$), promoting stability while permitting gradual evolution, consistent with Section 8's scaling factor tuning.
- **$R_\tau = R_k = 1.0$:** Measurement noise variances reflect expected variability in $z_\tau$ and $z_k$ due to proposal noise or attacks. A value of 1.0 assumes a standard deviation of 1 in normalized deviations, a reasonable default per Section 8's note, balancing responsiveness and smoothing.

#### 4.3.4. Step 4: Outlier Filtering with Hybrid Threshold Comparison

Proposals are classified as inliers ($\delta_v = 1$) or outliers ($\delta_v = 0$) using both sub-approaches, requiring both conditions to hold.

**Mean-Based Threshold Comparison**

1. Compute the deviation from the mean: $d_{v,\mu} = |S_v - \mu|$.
2. Compute the threshold: $T_\mu(t) = \tau(t) \cdot \sigma$, where $\tau(t)$ is the dynamic threshold from the Kalman filter.
3. Compare: Check if $d_{v,\mu} \leq T_\mu(t)$.
4. Result: If true, the proposal passes the mean-based check; if false, it's an outlier under this sub-approach.

**Median-Based Threshold Comparison**

1. Compute the deviation from the median: $d_{v,M} = |S_v - M|$ (already calculated for MAD).
2. Compute the threshold: $T_M(t) = k(t) \cdot \mathrm{MAD}_n$, where $k(t)$ is the dynamic threshold from the Kalman filter.
3. Compare: Check if $d_{v,M} \leq T_M(t)$.
4. Result: If true, the proposal passes the median-based check; if false, it's an outlier under this sub-approach.

**Hybrid Decision**

- Combine results:

$$\delta_v = \begin{cases} 1, & \text{if } d_{v,\mu} \leq \tau(t) \cdot \sigma \text{ and } d_{v,M} \leq k(t) \cdot \mathrm{MAD}_n, \\ 0, & \text{otherwise.} \end{cases}$$

- A proposal is an inlier only if it passes both checks, ensuring robustness (median) and sensitivity to variance (mean).

### 4.3.5. Step 5: Final Score Aggregation

Aggregate only inlier proposals:

$$S_{\text{agg}} = \frac{\sum_{v \in V_{\text{selected}}} \delta_v S_v}{\sum_{v \in V_{\text{selected}}} \delta_v}.$$

If no inliers exist ($\sum \delta_v = 0$), the protocol may revert to $C_W^{\text{old}}$ or flag the update for review, though typically $n$ is large enough (due to VRF selection) to ensure some inliers.

### 4.3.6. Rationale

The hybrid approach leverages the mean's ability to detect variance-based anomalies and the median's resistance to extreme values or collusion. Dynamic thresholds $\tau(t)$ and $k(t)$, updated via Kalman filters, adapt to network conditions—e.g., tightening during stability to reduce false positives and widening during attacks to catch subtle manipulations. This enhances COSINE's fraud detection (Section 6) and aligns with its self-tuning design (Section 8).

## 4.4. Phase 4: Performance Score Adjustments

A validator's *performance score* $P_v$ is updated after each aggregation:

$$P_v^{(t+1)} = \max\bigl(0, \min\bigl(1,\ P_v^{(t)} + \Delta_v\bigr)\bigr),$$

where

$$\Delta_v = \begin{cases} +\Delta^+, & \delta_v = 1, \\ -\Delta^-, & \delta_v = 0. \end{cases}$$

Repeated malicious deviations may also trigger stake slashing, providing strong incentives to propose correct updates.

## 4.5. Phase 5: Validator Reward Distribution

For each update event $i$, a total reward $R_{\text{total}}^{(i)}$ from the network reward pool is shared among the validators whose proposals were *inliers* ($\delta_v = 1$):

$$R_{v,i} = \begin{cases} \dfrac{R_{\text{total}}^{(i)}}{\sum_{v \in V_{\text{selected}}} \delta_v}, & \delta_v = 1, \\ 0, & \delta_v = 0. \end{cases}$$

Hence, validators that remain near the consensus mean for the new credit score receive compensation.

## 4.6. Cost Accumulation for Wallet Updates

Each wallet $W$ accumulates the total cost of validator rewards spent on its updates:

$$C_{\text{acc}}(W) = \sum_{i=1}^{n} R_{\text{total}}^{(i)},$$

where $n$ is the number of score update events since the wallet was last *verified.* Upon official verification, a *fee* is paid that covers this accumulated cost, as described in Section 12.

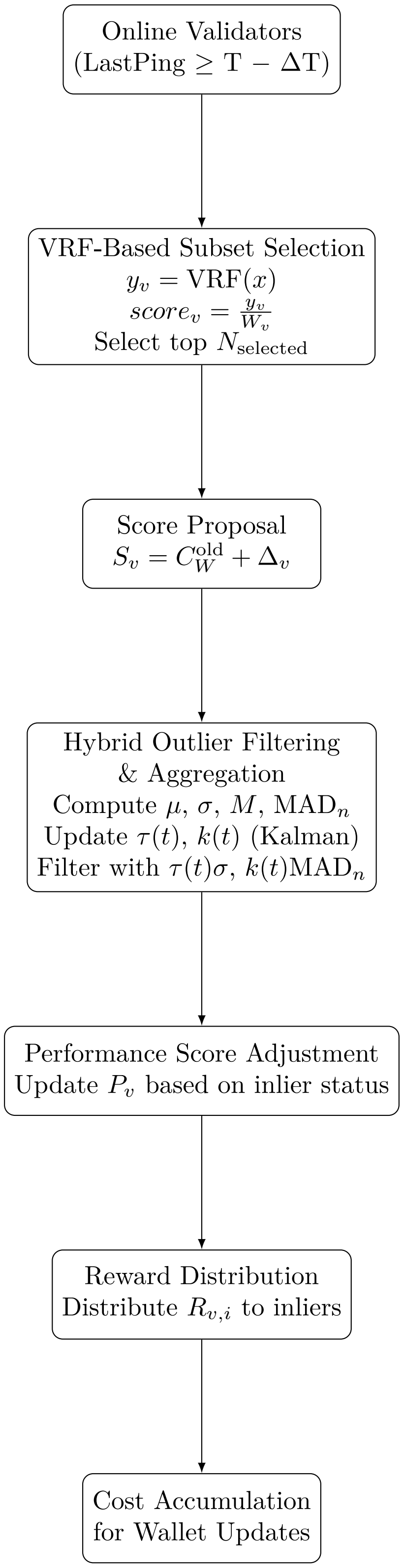
Online Validators
(LastPing $\geq$ T $-$ $\Delta$T)
VRF-Based Subset Selection
$y_v = \mathrm{VRF}(x)$
$score_v = \frac{y_v}{W_v}$
Select top $N_{\text{selected}}$
Score Proposal
$S_v = C_W^{\text{old}} + \Delta_v$
Hybrid Outlier Filtering
& Aggregation
Compute $\mu$, $\sigma$, $M$, $\mathrm{MAD}_n$
Update $\tau(t)$, $k(t)$ (Kalman)
Filter with $\tau(t)\sigma$, $k(t)\mathrm{MAD}_n$
Performance Score Adjustment
Update $P_v$ based on inlier status
Reward Distribution
Distribute $R_{v,i}$ to inliers
Cost Accumulation
for Wallet Updates

Figure 2: Mechanics of Validator Operations.

## 5. Decentralized Storage and Updates

The COSINE Layer-2 (L2) protocol requires a robust, decentralized, tamper-proof, and scalable storage system to underpin its real-time credit scoring, fraud mitigation, and cross-chain interoperability functionalities. This section details how COSINE achieves these objectives by maintaining all necessary data—wallet-specific records, global variables, transaction logs, and cross-chain bridging states—within a self-sufficient ledger managed exclusively by its validator network. COSINE integrates all operations, including bridging and updates, directly into its protocol, ensuring trust-minimization and operational independence. Here, we outline the data structures, update mechanisms, bridging processes, and scalability strategies, providing a comprehensive blueprint that aligns with our existing architecture (e.g., Sections 4, 8, and 10).

### 5.1. Objectives of Decentralized Storage

The COSINE L2 ledger must meet the following goals:

- **Decentralization:** All data is replicated across validators, avoiding reliance on external systems like IPFS or Layer-1 (L1) blockchains.
- **Tamper-Proofing:** Data integrity is cryptographically enforced, ensuring immutability of historical records and current states.
- **Scalability:** The system supports a growing number of wallets, transactions, and cross-chain interactions without compromising performance.
- **Self-Sufficiency:** All operations—storage, updates, and bridging—are managed internally by COSINE validators.
- **Accessibility:** Validators and light clients can efficiently query wallet data and global variables.

To achieve these, we define a ledger structure with specific data components, detailed below, and integrate update and bridging processes directly into the validator workflow.

### 5.2. COSINE L2 Ledger Structure

The COSINE L2 ledger is a blockchain-like structure where each block encapsulates the protocol's state and transaction history. It consists of three primary components:

1. **Block Header:** Contains metadata linking blocks and committing to the state:
   - Previous block hash: Ensures chain continuity (e.g., SHA-256 of the prior header).
   - Timestamp: Unix timestamp of block creation.
   - State Roots: Merkle roots for wallet state (`WST_root`), global state (`GSO_root`), and linked L1 mappings (`L1T_root`).
   - Validator Signatures: Multi-signature from the VRF-selected validator subset (Section 4.2).
2. **Transactions:** A list of events altering the state (e.g., token transfers, votes, linking requests, verification requests, bridge operations).
3. **State Updates:** Changes applied to the wallet state trie, global state object, or linked L1 trie post-consensus.

Blocks are produced periodically (e.g., every 5 minutes) or triggered by sufficient transaction volume, with validators using the hybrid mean-median approach (Section 4.3).

#### 5.2.1. Wallet State Trie (WST): Wallet-Specific Data

The Wallet State Trie (WST) is a Merkle Patricia Trie (MPT) storing data for each COSINE L2 wallet. An MPT combines the efficiency of a Patricia trie (compact key storage) with the cryptographic security of a Merkle tree, enabling fast updates and proofs of inclusion/exclusion. The trie is committed to via `WST_root` in each block header.

#### Key and Value Structure

- **Key:** The COSINE L2 wallet address (a 256-bit hash, e.g., `keccak256(public_key)`).
- **Value:** A serialized object containing all wallet-specific fields, structured as follows:

#### Fields in the Serialized Object

- **Linked L1 Addresses (`linked_L1_addresses`):**
  - Format: List of tuples `[(chain_id, address), ...]`.
  - `chain_id`: Integer identifying the blockchain (e.g., 1 for Ethereum, 501 for Solana).
  - `address`: L1 address (e.g., 160-bit Ethereum address, 256-bit Solana public key).

  - Example: `[(1, 0x1234...), (501, SolanaAddrXYZ...)]`.
  - Purpose: Tracks cross-chain identity for credit scoring and interoperability (Section 10).

- **Compounded Cost of Computation (`C_acc(W)`):**
  - Format: Numeric value (e.g., in wei-like units of COSINE tokens).
  - Purpose: Accumulates validator rewards for updates (Section 4.6), reset to 0 after verification fee payment.
  - Initial Value: 0.

- **Reputation Score (`R_U`):**
  - Format: Float in `[0, 1]`.
  - Purpose: Measures voting reliability (Section 7.4).
  - Initial Value: 0.1 (baseline for new wallets).

- **Rolling Statistics (`rolling_stats`):**
  - Format: Map of exponential moving averages (EMAs), e.g., `{"_V": {value, last_updated}, "_V": {value, last_updated}, ...}`.
  - Fields: `_V(t)`, `_V(t)` (vote statistics), `_R(t)`, `_R(t)` (association risk), etc.
  - `value`: Current EMA value (float).
  - `last_updated`: Unix timestamp or block height of last update.
  - Purpose: Supports dynamic scaling (Section 8).
  - Initial Values: Seeded with defaults (e.g., `_V = 0`, `_V = 1`).

- **Timestamp of Last Update (`last_updated`):**
  - Format: Unix timestamp or block height.
  - Purpose: Tracks data recency for validation and synchronization.
  - Initial Value: Set at wallet creation.

- **Credit Score (`C_W`):**
  - Format: Unbounded float or integer.
  - Purpose: Reflects creditworthiness (Section 8).
  - Initial Value: 0 (default for new wallets).

- **Association Risk Score (`R_assoc(W)`):**
  - Format: Numeric value (float).
  - Purpose: Sums transaction weights from suspicious links (Section 6.1).
  - Initial Value: 0.

- **Vote History (`vote_history`):**
  - Format: List of vote events, e.g., `[voter_address, vote_value, timestamp, effective_vote, ...]`.

  - `voter_address`: Address of the voting wallet.
  - `vote_value`: `-1` or `+1`.
  - `timestamp`: When the vote was cast.
  - `effective_vote`: `R_U * vote_value`.
  - Purpose: Tracks voting impacts for auditing and reputation updates.

- **Token Balance (`token_balance`):**
  - Format: Numeric value (e.g., in smallest COSINE token units).
  - Purpose: Tracks COSINE token holdings on L2.
  - Initial Value: 0, unless initialized with tokens.

### 5.2.2. Global State Object (GSO): Protocol-Wide Variables

The Global State Object (GSO) stores variables applicable to the entire protocol, committed to via `GSO_root` (e.g., a SHA-256 hash of the serialized object) in each block header.

#### Fields in the GSO

- **Scaling Factors (`scaling_factors`):**
  - Format: Map, e.g., `{"K_neg": {value, P_t}, "K_pos": {value, P_t}, "K_assoc": {value, P_t}, "K_rehab": {value, P_t}}`.
  - `value`: Current scaling factor (float).
  - `P_t`: Kalman filter error variance for dynamic updates (Section 8).
  - Initial Values: All set to 1.0, `P_t = 1`.
  - Purpose: Adjusts credit score shifts dynamically.
- **Network Reward Pool (`network_reward_pool`):**
  - Format: Numeric value (COSINE token units).
  - Initial Value: 400,000,000 (40% of total supply, Section 13).
  - Purpose: Funds validator rewards (Section 4.5).
- **Total Token Supply on L2 (`total_token_supply_L2`):**
  - Format: Numeric value (COSINE token units).
  - Initial Value: Adjusted post-initial distribution (e.g., 600,000,000 after allocating 40% to rewards).
  - Purpose: Tracks circulating supply on L2.
- **Validator Stakes and Performance Scores (`validators`):**
  - Format: Map, e.g., `{v_addr: {"stake": S_v, "performance": P_v, "last_ping": timestamp}}`.

  - `stake`: Staked tokens (minimum 100,000, Section 13).
  - `performance`: Float in `[0, 1]` (Section 4.4).
  - `last_ping`: Timestamp of last ping (Section 4).
  - Initial Values: Set for genesis validators.
  - Purpose: Tracks validator eligibility and influence.

- **Linked L1 Mapping (`association_mapping`):**
  - Format: Map, e.g., `{(chain_id, L1_address): L2_wallet_address}`.
  - Purpose: Enforces one-to-one L1-to-L2 linking (Section 10).
  - Initial Value: Empty.

- **Bridged Token States (`bridged_tokens`):**
  - Format: Map, e.g., `{chain_id: locked_amount}`.
  - Example: `{1: 10,000,000}` (10M tokens locked for Ethereum).
  - Purpose: Tracks tokens bridged to L1 chains.
  - Initial Value: Empty.

#### 5.2.3. Linked L1 Trie (L1T): Enforcing One-to-One Linking

The Linked L1 Trie (L1T) ensures each L1 address maps to exactly one L2 wallet, committed to via `L1T_root`.

- **Key:** Hash of `(chain_id, L1_address)` (e.g., `keccak256(chain_id || L1_address)`).
- **Value:** The associated L2 wallet address.
- Purpose: Prevents malicious re-linking (Workflow 7, Section 11).

### 5.3. Transaction Types and Recording

The ledger records all transactions that alter state:

- **Token Transfers:** `{sender, receiver, amount, timestamp}`.
- **Vote Events:** `{voter, target_wallet, vote_value (-1 or +1), timestamp, captcha_solution}`.
- **Linking Requests:** `{L2_wallet, chain_id, L1_address, signature}`.
- **Verification Requests:** `{wallet, requester, threshold, fee_paid}`.
- **Bridge Operations:** `{operation (lock/mint or burn/redeem), chain_id, amount, user}`.

Each transaction is included in a block, processed by validators, and reflected in state updates.

### 5.4. Update Mechanisms

Updates to the WST, GSO, and L1T occur sequentially via validator consensus, triggered by transactions or events (e.g., votes, association penalties).

#### 5.4.1. Wallet State Updates

1. **Event Trigger:** A transaction (e.g., vote, token transfer, linking request) targets wallet $W$.
2. **Validator Selection:** A subset $V_{\text{selected}}$ is chosen via VRF (Section 4.2).
3. **Computation:** Validators compute updates (e.g., $\Delta_{\text{vote}}$, $\Delta_{\text{assoc}}$) using GSO scaling factors.
4. **Consensus:** Proposals are aggregated with outlier filtering (Section 4.3), yielding a final value (e.g., $C_W$).
5. **State Modification:** Update the WST entry for $W$ (e.g., set `C_W = S_agg`, increment `C_acc(W)`).
6. **Commitment:** New `WST_root` is computed and included in the block header.

#### 5.4.2. Global State Updates

- **Scaling Factors:** Updated via Kalman filters after each event (Section 8).
- **Reward Pool:** Decreased by $R^{(i)}_{\text{total}}$ per update (Section 4.5).
- **Validator Data:** Adjusted for stakes and performance (Section 4.4).
- **Process:** Validators propose GSO changes, reach consensus, and commit the new `GSO_root`.

#### 5.4.3. VRF Implementation for Trustless Selection

The Verifiable Random Function (VRF) ensures trustless validator selection:

1. **Seed Generation:** Use the previous block hash as seed $x$ (e.g., `SHA-256(header`$_{t-1}$`))`.**`VRF Computation:`**$Eac$
   `computes:`

$$y_v = \texttt{VRF}_{sk_v}(x), \quad \pi_v = \texttt{Proof}_{sk_v}(x),$$

   `using a standard VRF (e.g., based on elliptic curve cryptography).`

3. **`Score Calculation:`**

$$W_v = S_v^{\alpha} \times (1 + \beta P_v), \quad \texttt{score}_v = y_v / W_v,$$

   `where` $\alpha = 0.5$, $\beta = 0.2$ `(configurable in the Global State Object (GSO)).`

4. `Verification:` `Validators broadcast` $\{y_v, \pi_v\}$, `peers verify` $\pi_v$ `against` $v$`'s public key and` $x$.
5. `Selection:` `Top` $N_{\texttt{selected}} = \lceil \gamma |V_{\texttt{online}}| \rceil$ `validators with smallest` $\texttt{score}_v$ `are chosen (`$\gamma = 0.1$`,` $N_{\max} = 100$`).`

This ensures randomness, verifiability, and decentralization without external dependencies.

## 5.5. Cross-Chain Bridging Mechanism

Since COSINE lacks smart contracts, bridging is integrated into the protocol and managed by validators, maintaining a total supply invariant of 1 billion tokens.

### 5.5.1. Bridging from COSINE L2 to L1

1. **User Request:** Submit `{lock/mint, chain_id, amount, L1_address}`.
2. **Validation:** Validators check `token_balance` $\geq$ `amount` in WST.
3. **Locking:**
   - Decrease `token_balance` of user's wallet by `amount`.
   - Increase `token_balance` of a bridge account (a reserved WST entry) by `amount`.
4. **Proof Generation:** Validators create a proof of lock:

$$\text{Proof} = \{\text{L2\_wallet}, \text{amount}, \text{chain\_id}, \text{tx\_id}, \text{Merkle\_proof}, \text{signatures}\}.$$

5. **Cross-Chain Relay:** Proof is relayed to the L1 chain (e.g., via an oracle or messaging protocol).
6. **Minting:** L1 bridge contract verifies proof and mints wrapped tokens (e.g., wCOSINE).
7. **State Update:** Increase `bridged_tokens[chain_id]` in GSO by `amount`.

### 5.5.2. Bridging from L1 to COSINE L2

1. **Burn on L1:** User burns wrapped tokens, generating a proof of burn (e.g., L1 tx hash).
2. **Submission:** Submit proof to COSINE L2 via transaction.
3. **Verification:** Validators confirm burn against L1 state.
4. **Unlocking:**
   - Decrease bridge account's `token_balance` by `amount`.
   - Increase user's `token_balance` by `amount`.
5. **State Update:** Decrease `bridged_tokens[chain_id]` in GSO by `amount`.

### 5.5.3. Supply Invariant

$$\text{total\_token\_supply\_L2} + \sum_{\text{chain\_id}} \text{bridged\_tokens[chain\_id]} = 1,000,000,000.$$

This is enforced by validators during every bridge operation.

## 5.6. Scalability and Optimization

- **Sharding:** If wallet count exceeds a threshold (e.g., 50M), shard WST by address prefix, assigning validators to shards.
- **Pruning:** Remove `vote_history` entries older than 5 years, archiving them off-chain for auditing.
- **Light Clients:** Provide Merkle proofs for querying `C_W`, `token_balance`, etc., reducing full node load.

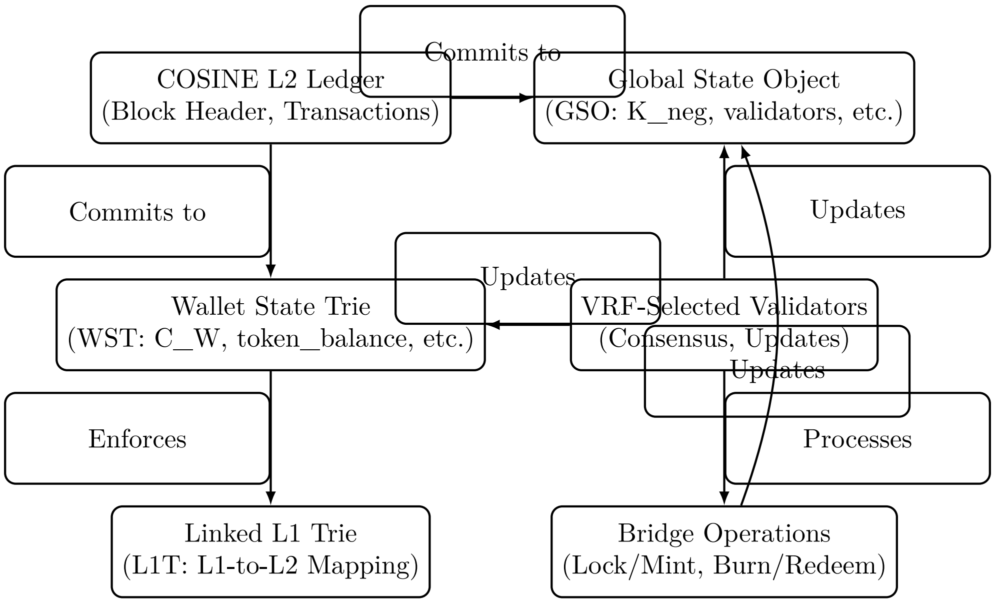
Commits to
COSINE L2 Ledger (Block Header, Transactions)
Global State Object (GSO: K_neg, validators, etc.)
Commits to
Updates
Updates
Wallet State Trie (WST: C_W, token_balance, etc.)
VRF-Selected Validators (Consensus, Updates)
Updates
Enforces
Processes
Linked L1 Trie (L1T: L1-to-L2 Mapping)
Bridge Operations (Lock/Mint, Burn/Redeem)

Figure 3: COSINE L2 Storage and Update Workflow

## 5.7. Remarks

The COSINE L2 ledger, with its WST, GSO, and L1T, provides a decentralized, tamper-proof, and scalable storage solution. Wallet-specific data enables real-time credit scoring and cross-chain linking, while global variables support protocol-wide operations. Integrated bridging ensures token interoperability without external dependencies, and validator-driven updates maintain self-sufficiency. This design aligns with COSINE's trust-minimized ethos, offering a robust foundation for its DeFi ecosystem.

## 6. Fraud Detection & Network-Wide Penalization

Fraud detection in COSINE leverages **randomized time windows** and **multi-hop association analysis** to penalize wallets linked to malicious activities. Once a wallet is flagged or if it accumulates suspicious on-chain interactions, partial penalties degrade its credit score. This section also describes how partial blacklisting works without whitelisting and how the dynamic scaling approach applies to association penalties.

### 6.1. Association Risk Analysis

#### 6.1.1. Random Time & Hop Limits: Rationale

To combat strategic laundering, COSINE uses:

- A **random time window** $T_{\text{limit}}$ sampled from a uniform range, e.g. 30 to 730 days after a triggering event. This means the protocol can examine transaction flows within that future interval.
- A **random hop limit** $H_{\text{limit}}$ in $\{1, \ldots, 15\}$. Any wallet that receives funds from a malicious address or from an intermediate address within $H_{\text{limit}}$ hops may be flagged for partial penalty.

Attackers thus cannot reliably predict how far or how long the system will look for suspicious links, forcing them to remain uncertain about attempts to route funds through multiple intermediary addresses.

#### 6.1.2. Measuring Transaction Data for Association

Every transaction $i$ from wallet $W_1$ to $W_2$ records:

$$A_i \text{ (amount)}, \quad t_i \text{ (timestamp)}, \quad n_{\text{cluster}} \text{ (clustering count)}.$$

A weight function:

$$w_i = \log(1 + A_i) \times [1 + \beta\,(n_{\text{cluster}} - 1)] \times e^{-\delta(t_{\text{current}} - t_i)},$$

captures the impact of transaction size, frequency, and recency:

- $\log(1 + A_i)$ models diminishing returns for large amounts,
- $1 + \beta(n_{\text{cluster}} - 1)$ accounts for multiple transactions in short intervals,
- $e^{-\delta(t_{\text{current}} - t_i)}$ decays older transactions' influence.

#### 6.1.3. Computing Association Risk

For wallet $W$, define

$$R_{\text{assoc}}(W) \;=\; \sum_{i \in \mathcal{T}(W,\, H_{\text{limit}},\, T_{\text{limit}})} w_i,$$

where $\mathcal{T}(W,\, H_{\text{limit}},\, T_{\text{limit}})$ is the set of transactions that link $W$ to a malicious wallet (directly or within $H_{\text{limit}}$ hops) during the relevant random time window. If $R_{\text{assoc}}(W)$ significantly exceeds typical network norms, a partial penalty is applied.

### 6.2. Dynamic Association Penalty: Self-Tuning Approach

COSINE tracks rolling means and standard deviations for association risk:

$$\mu_R(t), \;\; \sigma_R(t).$$

Whenever $R_{\text{assoc}}(W)$ is higher than $\mu_R(t)$ by a margin that exceeds a threshold multiple of $\sigma_R(t)$, a negative shift is triggered:

$$\Delta_{\text{assoc}} = -\, K_{\text{assoc}}(t) \left( \frac{(R_{\text{assoc}}(W) - \mu_R(t))_+}{\sigma_R(t)} \right)^{\gamma},$$

where $\gamma$ is an exponent (often 1 for linear effect, though it can be greater than 1), and $K_{\text{assoc}}(t)$ is a dynamically updated scaling factor. This factor evolves via a *feedback loop* (Section 8) that compares observed association penalties with expected standardized deviations.

### 6.3. Partial Blacklisting Mechanics

When $\Delta_{\text{assoc}} < 0$ is imposed, we call it a *partial penalty* or partial blacklisting. The wallet's score is not necessarily driven to extremely negative territory in one step, but repeated suspicious associations can accumulate and effectively block the wallet from typical lending thresholds. Importantly:

- **Handling Exchange wallets:** Exchange wallets, despite being high-volume, are protected from accidental partial blacklisting by their transactional volume and frequency—which is factored into the system's association weight calculation. As exchanges frequently interact with many wallets, their patterns are considered "normal" based on the cluster size and transaction timing, preventing them from being penalized unless clear evidence of malicious behavior emerges.
- **Voting Restrictions for Collusion Prevention:** If wallet $W$ is partially penalized for association with a malicious wallet $M$, then $W$ cannot participate in any subsequent votes to restore or rehabilitate $M$. This prevents a mutual absolution scenario.

### 6.4. Scams & Suspicious Patterns

Typical scams (e.g., rug pulls, pig butchering, exploit addresses) can be quickly flagged by the community, imposing a negative vote shift on the suspect wallet. Meanwhile, all wallets that continue interacting with the suspect wallet (directly or through multiple hops) risk partial penalties if the amounts, frequencies, or recency patterns exceed normal network activity. Over time, the system thus “chokes out” malicious clusters and their spin-off addresses.

## 7. Community Governance & Voting Mechanics

Community governance in COSINE revolves around *sequential, reputation-weighted vote events.* There are no fixed intervals or single "all-or-nothing" votes. Instead, each user can cast a *discrete* negative or positive vote at any time, and the effect on a suspect or target wallet's score is computed immediately. Over many votes, a final consensus emerges about whether a wallet is malicious, benign, or rehabilitated.

### 7.1. Voter Reputation ($R_U$)

Each voting wallet $U$ has a reputation score $R_U \in [0, 1]$. Initially, new wallets may have $R_U = 0$ or a small baseline (e.g. 0.1) to prevent purely zero influence. As $U$ votes consistently with the evolving consensus, $R_U$ increases; if $U$ repeatedly votes contrary to the eventual outcome, $R_U$ falls, limiting future influence.

### 7.2. Sequential Vote Events

#### 7.2.1. Event Data

When a user $U$ casts a vote $v_U \in \{-1, +1\}$ at block timestamp $t_v$:

- The system records $\text{EffectiveVote}_U = R_U \times v_U$.
- This value is compared to the current rolling average $\mu_V(t)$ of effective votes, as well as the rolling standard deviation $\sigma_V(t)$.

#### 7.2.2. Deviation $\Delta V$

We define:

$$\Delta V = (R_U \cdot v_U) \; - \; \mu_V(t).$$

If $\Delta V$ is only slightly outside typical historical norms, it might not move the target wallet's score much. If $\Delta V$ is large (for instance, many high-reputation users strongly voting negative or positive against the average), the magnitude of the resulting shift is bigger.

### 7.2.3. Dynamic Vote Impact

The credit score shift from a single vote event is:

$$\Delta_{\text{vote}} = \begin{cases} -K_{\text{neg}}(t)\,\big(\frac{|\Delta V|}{\sigma_V(t)}\big)^{\gamma}, & \text{if } \Delta V < -\lambda\,\sigma_V(t), \\ +K_{\text{pos}}(t)\,\big(\frac{|\Delta V|}{\sigma_V(t)}\big)^{\gamma}, & \text{if } \Delta V > +\lambda\,\sigma_V(t), \\ 0, & \text{otherwise.} \end{cases}$$

Here:

- $\lambda$ is a tolerance factor (e.g., 2.5) controlling how many standard deviations away from the mean is needed to trigger a non-zero shift.
- $\gamma$ is an exponent controlling how quickly the penalty or reward grows relative to standardized deviation.
- $K_{\text{neg}}(t)$ and $K_{\text{pos}}(t)$ are self-tuning scaling factors, each updated over time using a Kalman filter-based feedback loop (Section 8).

If $\Delta V$ is within $\pm\lambda\,\sigma_V(t)$, the system deems it "near average" and does not shift the wallet's score.

### 7.3. Positive vs. Negative Vote Accumulation

Unlike older major/minor penalty designs, COSINE aggregates *all* negative or positive votes over time:

- **Repeated Negative Votes:** If many high-reputation users strongly cast negative votes ($v_U = -1$), the target wallet can quickly be driven into a lower credit bracket, effectively blacklisting it from undercollateralized lending.
- **Positive Votes & Rehabilitation:** If the wallet's situation improves or if the community believes the negative votes were in error, a wave of positive votes ($v_U = +1$) can incrementally restore the wallet's score.

This continuous, real-time approach allows the network's stance on a wallet to evolve organically, with no single final "yes/no" cut-off.

## 7.4. Reputation Updates for Voters

After each vote, or after a short delay so the system can gauge the "direction" of subsequent votes, the protocol updates $R_U$. A typical approach:

$$R_U^{(t+1)} = \text{clamp}\Big(R_U^{(t)} + \alpha_R \cdot (1 - \epsilon_U) - \beta_R \cdot \epsilon_U,\ 0,\ 1\Big),$$

where:

- $\epsilon_U$ measures how far the user's vote $\text{EffectiveVote}_U$ ended up from the final, longer-term net direction.
- $\alpha_R, \beta_R$ are learning rates, controlling how quickly $R_U$ rises or falls.

Hence, consistent alignment with the evolving consensus yields gradually higher reputation, while repeated misalignment reduces it.

## 7.5. Ineligibility for Associated Wallets

Recall that if a wallet $W$ is penalized for association with a malicious wallet $M$, it cannot vote on $M$'s malicious status or rehabilitation. This rule halts direct collusion, where malicious wallet clusters might artificially inflate each other's reputation or absolve each other from penalty.

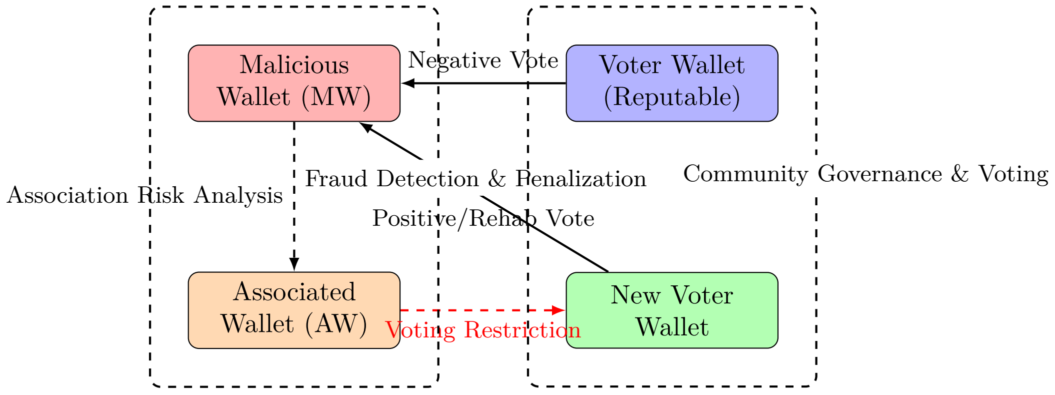

Figure 4: Overview of Fraud Detection, Network-Wide Penalization, and Community Governance & Voting Mechanics.

## 8. Credit Score Updates: Single-Dimensional Scalar Approach

COSINE tracks a single-dimensional credit score $C_W$ for each wallet. That score is updated incrementally according to the negative or positive votes from Section 7 and the partial association penalties from Section 6. Additionally, a positive rehabilitation "vote wave" can push $C_W$ upward if the community sees evidence the wallet has reformed.

### 8.1. Unified Update Equation

At each relevant event—i.e., a new vote or an association penalty—$\Delta_{\text{vote}}$ or $\Delta_{\text{assoc}}$ is computed. The *continuous* credit score update is:

$$C_W^{(t+1)} \;=\; C_W^{(t)} \;+\; \Delta_{\text{vote}} \;+\; \Delta_{\text{assoc}} \;+\; \Delta_{\text{rehab}},$$

where $\Delta_{\text{rehab}}$ is the shift from any *explicit rehabilitation mechanism* described next.

### 8.2. Rehabilitation Updates

Even if a wallet is severely penalized by negative votes or suspicious interactions, it can improve its score through:

- **Positive Vote Events:** Over time, if new high-reputation voters strongly cast $v_U = +1$ for the wallet, the normal $\Delta_{\text{vote}}$ can push it upward again.
- **Explicit Rehabilitation Votes:** In some versions of the protocol, the community may declare a separate "rehab" vote. This is effectively a strongly positive wave. Suppose a net rehabilitation measure $V_{\text{rehab}} = \sum_{\text{Eligible}} R_U \cdot (+1)$ arises, significantly above the historical rolling mean. Then:

  $$\Delta_{\text{rehab}} \;=\; +K_{\text{rehab}}(t)\,\left(\frac{(V_{\text{rehab}} - \mu_{R,V}(t))_+}{\sigma_{R,V}(t)}\right)^{\gamma},$$

  where $\mu_{R,V}(t)$ and $\sigma_{R,V}(t)$ track the typical magnitude of positive "rehab" signals. The scaling factor $K_{\text{rehab}}(t)$ is updated via the same feedback loop approach described below.

## 8.3. Dynamic Scaling Factors: Feedback Loops

**Overview** Each type of event (negative vote, positive vote, association penalty, rehabilitation) has an associated scaling factor:

$$K_{\text{neg}}(t), \quad K_{\text{pos}}(t), \quad K_{\text{assoc}}(t), \quad K_{\text{rehab}}(t).$$

These factors start at initial values **(e.g., 1.0)** and self-tune using a Kalman filter to dynamically adjust based on the discrepancy between observed and expected impacts. This ensures that penalties and rewards remain appropriately scaled and responsive to current network conditions, leveraging the Kalman filter's ability to adapt quickly and handle noisy data effectively.

**Kalman Filter Update Process** For each scaling factor, such as $K_{\text{neg}}(t)$ for negative votes, we employ a Kalman filter where the scaling factor is treated as a state variable. The update occurs sequentially with each relevant event (e.g., a negative vote). The process is as follows:

1. **State Model:** The scaling factor is modeled with a simple random walk to allow adaptation over time:
$$K_{\text{neg}}(t) = K_{\text{neg}}(t-1) + w_t,$$
where $w_t \sim N(0, Q)$ is the process noise with variance $Q$, reflecting expected variation in the scaling factor.

2. **Measure Observed and Expected Impacts:**
   - **ObservedImpact:** For each negative vote event $i$, the actual shift applied is:
   $$\Delta_{\text{vote}}^{\text{obs}}(i) = -K_{\text{neg}}(t) \cdot \left( \frac{|\Delta V(i)|}{\sigma_V(t)} \right)^{\gamma},$$
   where $\Delta V(i) = (R_U \cdot v_U) - \mu_V(t)$. We define $\text{ObservedImpact}(t) = |\Delta_{\text{vote}}^{\text{obs}}(i)|$.
   - **ExpectedImpact:** The expected shift is based on the prior scaling factor:
   $$\Delta_{\text{vote}}^{\text{exp}}(t) = K_{\text{neg}}(t-1) \cdot \left( \frac{|\Delta V(i)|}{\sigma_V(t)} \right)^{\gamma},$$
   and $\text{ExpectedImpact}(t) = |\Delta_{\text{vote}}^{\text{exp}}(t)|$.

3. **Kalman Filter Update:** The scaling factor is updated using the Kalman filter equation:

$$K_{\text{neg}}(t+1) = K_{\text{neg}}(t) + G_t \cdot (\text{ObservedImpact}(t) - \text{ExpectedImpact}(t)),$$

   where $G_t$ is the Kalman gain, computed as:

$$G_t = \frac{P_t}{P_t + R},$$

   - $P_t$ is the estimated error variance of $K_{\text{neg}}(t)$ before the update, predicted as $P_t = P_{t-1} + Q$,
   - $R$ is the measurement noise variance, reflecting uncertainty in the observed impacts,
   - After the update, $P_{t+1} = (1 - G_t) \cdot P_t$.

4. **Interpretation:**
   - If ObservedImpact $>$ ExpectedImpact, the shift was larger than anticipated, and $K_{\text{neg}}$ increases to align future shifts with observed effects.
   - If ObservedImpact $<$ ExpectedImpact, the shift was smaller, and $K_{\text{neg}}$ decreases.
   - The Kalman gain $G_t$ balances the influence of new measurements against the prior estimate, adapting quickly to significant changes while filtering noise.

**Application to All Scaling Factors** The same Kalman filter approach applies to $K_{\text{pos}}(t)$, $K_{\text{assoc}}(t)$, and $K_{\text{rehab}}(t)$, with ObservedImpact and ExpectedImpact defined based on their respective shifts ($\Delta_{\text{vote}}$ for positive votes, $\Delta_{\text{assoc}}$ for association penalties, $\Delta_{\text{rehab}}$ for rehabilitation). Each uses separate process noise $Q$ and measurement noise $R$ parameters, tuned to the event type.

**Recommended Default Values for $Q$ and $R$:**

In the Kalman filter used to update scaling factors (e.g., $K_{\text{neg}}, K_{\text{pos}}, K_{\text{assoc}}, K_{\text{rehab}}$), two key parameters govern the adaptability and stability of the system:

- **$Q$ (Process Noise Variance):** This parameter represents the expected variance in the scaling factor between updates. A larger $Q$ allows the scaling factor to adapt more quickly to changes but may introduce sensitivity to noise. A default value of $Q = 0.01$ is recommended, assuming moderate variation in scaling factors (e.g., typical changes of around 0.1 per update). This ensures gradual adaptation while maintaining stability.

- **$R$ (Measurement Noise Variance):** This parameter reflects the uncertainty or noise in the observed impacts (e.g., differences between observed and expected credit score shifts). A larger $R$ reduces the filter's reliance on noisy measurements, promoting smoother updates. A default value of $R = 1.0$ is suggested, assuming a standard deviation of 1.0 in the measurement noise, which is a common starting point when specific noise data is unavailable.

- **Balance Between Responsiveness and Stability:** With $Q = 0.01$ and $R = 1.0$, the filter prioritizes stability, preventing overreaction to isolated events while allowing adaptation to persistent trends. These defaults are suitable for all scaling factors but can be adjusted based on empirical data.

**Implementation Note** The specific values of $Q$ and $R$ are configurable protocol parameters, determined empirically to optimize responsiveness and stability. This mechanism offers adaptability to sudden network shifts and robustness against noisy data.

### 8.4. No Upper or Lower Bound on $C_W$

Credit scores can become very large (for trustworthy wallets) or very negative (for repeated offenders). Different DeFi protocols can place their own acceptance thresholds. Because COSINE is single-dimensional, integrators simply interpret the final numeric value. Alternatively, they can rely on the *cosine similarity* check (Section 10) for acceptance or rejection.

### 8.5. Embedding for Cosine Similarity Verification

Instead of directly returning $C_W$, the protocol produces a two-dimensional vector representing the wallet's *normalized* credit score:

$$\hat{C}_W = \frac{C_W - \mu_C}{\sigma_C} \quad \text{(normalization)},$$

where $\mu_C$ and $\sigma_C$ are rolling statistics for all wallets' credit scores. The wallet's vector might be

$$\mathbf{v}_W = \begin{pmatrix} \hat{C}_W \\ 1 \end{pmatrix}.$$

If a user wants to see if $W$ is "close to" a threshold $T$, the threshold is also normalized:

$$\hat{T} = \frac{T - \mu_C}{\sigma_C}, \quad \mathbf{v}_T = \begin{pmatrix} \hat{T} \\ 1 \end{pmatrix}.$$

Then

$$\text{cosine\_sim}(\mathbf{v}_W, \mathbf{v}_T) \; = \; \frac{\mathbf{v}_W \cdot \mathbf{v}_T}{\|\mathbf{v}_W\| \, \|\mathbf{v}_T\|}.$$

A similarity near 1 indicates $C_W$ is effectively at or above the threshold. This approach mitigates raw score volatility and provides a stable measure of "closeness" to a desired credit level. Equally, a user can compare $\mathbf{v}_W$ to another wallet's vector $\mathbf{v}_{W'}$ to see how similar their credit statuses are. If cosine_sim$(\mathbf{v}_W, \mathbf{v}_{W'})$ is high, the two wallets share nearly the same credit standing.

## 9. Vote Validation

In the COSINE protocol, voting is a fundamental component of community governance and credit scoring, enabling users to provide feedback on wallet trustworthiness. To ensure the integrity of this process and prevent Sybil attacks—where malicious actors create multiple identities to manipulate outcomes—we implement a decentralized CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) system for vote validation. This section outlines the voting process, the rationale for choosing decentralized CAPTCHAs, and the detailed mechanism by which votes are validated, ensuring compatibility with the existing validator architecture for credit score updates.

### 9.1. Rationale for Decentralized CAPTCHAs

The primary challenge in designing a voting system for the COSINE protocol is ensuring that votes are cast by genuine human users, not automated bots, while maintaining accessibility in real-world scenarios. Traditional decentralized voting systems often rely on staking mechanisms, where users lock up tokens to participate. However, this approach is impractical for COSINE users, particularly those who have lost funds due to malicious transactions. Expecting victims of scams to pay or stake tokens to vote on the trustworthiness of a wallet would create a significant barrier to entry, reducing participation and undermining the protocol's effectiveness.

Decentralized CAPTCHAs address this issue by providing a lightweight, cost-free method to verify human participation. Unlike staking, CAPTCHAs impose no financial burden on users, making voting accessible even to those who have suffered losses. Furthermore, as seen in the in the Post-Transaction Feedback in section 11.5.3 below, users who have verified a credit score and transacted—either on the COSINE Layer 2 or a Layer 1 blockchain—are required to vote before performing their next verification. This positions voting as a critical step, necessitating a secure yet user-friendly validation mechanism.

### 9.2. Decentralized CAPTCHA Verification Process

The COSINE protocol integrates CAPTCHA verification into its existing validator framework, leveraging a subset of validators selected via a Verifiable Random Function (VRF) to ensure decentralization. Importantly, to optimize user experience, each vote requires the user to solve only a single CAPTCHA, presented immediately after casting their vote. This timing aligns with the psychological flow of task completion, reducing friction. Below, we detail the step-by-step process of vote validation.

1. **Vote Submission and Validator Selection**
   - When a user $U$ casts a vote $v_U \in \{-1, +1\}$ for a target wallet $W$, the protocol triggers the validation process.
   - A subset of validators $V_{\text{selected}}$ is chosen via VRF, based on their stake and performance scores (as outlined in Section 4.2).

- From $V_{\text{selected}}$, one validator $v_{\text{gen}}$ is randomly selected to generate the CAPTCHA.

2. **CAPTCHA Generation and Presentation**
   - $v_{\text{gen}}$ generates a CAPTCHA challenge $\text{CAPTCHA}_{\text{challenge}}$ (e.g., an image selection task) and determines the correct solution $\text{solution}_{\text{correct}}$.
   - $v_{\text{gen}}$ sends $\text{CAPTCHA}_{\text{challenge}}$ to the user $U$ and shares $\text{solution}_{\text{correct}}$ with the other validators in $V_{\text{selected}}$.
3. **User Solution Submission**
   - The user $U$ solves the CAPTCHA and submits their solution $\text{solution}_U$ via their client (e.g., a wallet or web app).
4. **Decentralized Verification**
   - Each validator in $V_{\text{selected}}$ independently checks if $\text{solution}_U = \text{solution}_{\text{correct}}$.
   - Define $\delta_v$ as the decision of validator $v$:

$$\delta_v = \begin{cases} 1, & \text{if } \text{solution}_U = \text{solution}_{\text{correct}}, \\ 0, & \text{otherwise.} \end{cases}$$

   - The vote is accepted if a majority of validators agree:

$$\text{Vote Accepted} = \begin{cases} 1, & \text{if } \sum_{v \in V_{\text{selected}}} \delta_v > \frac{|V_{\text{selected}}|}{2}, \\ 0, & \text{otherwise.} \end{cases}$$

5. **Credit Score Update**
   - If the vote is accepted, the same subset $V_{\text{selected}}$ proceeds to compute the credit score update for wallet $W$ based on $v_U$ (see Section 8).
   - The computational cost of CAPTCHA verification is factored into the overall cost of updating the credit score after this vote-specific event, ensuring efficiency.
6. **Reputation Update**
   - The reputation $R_U$ of user $U$ is updated accordingly (see Section 7.4).

## 9.3. User Experience and CAPTCHA Design

For an optimal user experience, the CAPTCHA is presented immediately after the user submits their vote, requiring only one challenge per vote. CAPTCHAs often analyze *how* they are solved (e.g., via mouse movements or timing) to distinguish humans from bots.

## 9.4. Security and Validator Incentives

Validators are incentivized to act honestly through staking and performance-based rewards. The random selection of $v_{\text{gen}}$ via VRF, combined with the majority-vote requirement, minimizes the risk of manipulation by any single validator. If $v_{\text{gen}}$ provides an incorrect $\text{solution}_{\text{correct}}$, honest validators in $V_{\text{selected}}$ would reject the user's solution, preventing the vote from being accepted. This alignment of incentives ensures the integrity of the CAPTCHA verification process.

## 9.5. Integration with Credit Score Updates

The decentralized CAPTCHA system is fully compatible with the validator mechanism for credit score updates. Since $V_{\text{selected}}$ is already tasked with computing the credit score update after a vote, extending their role to include CAPTCHA verification is a natural fit. This dual-purpose approach avoids additional validator selection overhead and incorporates the CAPTCHA validation cost into the broader computational effort of processing a vote-specific event, distinct from partial association events.

## 10. Cross-Chain Wallet Verification and Associated Credit Scoring

COSINE facilitates a single credit score across multiple L1 or sidechains by letting users *link* external addresses to the L2 wallet. The protocol then applies the same association analysis (random time windows, multi-hop checks) to suspicious transactions on external chains, as well as enabling cross-chain verification of the wallet's credit "similarity."

1. **Verify** that a given L1 address is indeed owned by (or under control of) the same individual or entity controlling an L2 wallet on COSINE.
2. **Ensure** that each L1 address is linked to only one COSINE L2 wallet (to prevent malicious users from associating the same L1 address with multiple COSINE addresses).
3. **Monitor** relevant L1 transactions to update credit scores accordingly (partial penalties, rehabilitations, etc.).
4. **Require** external lenders (or counterparties on L1) to provide minimal feedback ( *Yes/No* and *Trust/No-Trust* votes) when they transact directly on L1 with a COSINE-associated user, further enhancing the credit score reliability in a multi-chain context.

### 10.1. Linking External Addresses to an L2 Wallet

- **Proof of Ownership:** The user signs a challenge with the private key of the L1 address to confirm they control it.
- **One-to-One Linking:** An L1 address can only be linked to one L2 wallet.
- **Association Risk Import:** Once linked, any suspicious or malicious interactions that occur (or have occurred in the random time window) on the external chain will feed into $R_{\text{assoc}}(W^{(2)})$.

### 10.2. Cross-Chain Malicious Evader Scenario

An attacker might attempt to move funds from a blacklisted L1 address to a fresh L1 address, then link that new address to a new L2 wallet. However, the protocol's random time window and multi-hop tracing can detect the suspicious connection. The newly linked wallet thus receives a partial penalty based on the measured association risk. This strongly discourages reputation laundering across chains.

### 10.3. Cosine Similarity Verification Across Chains

A user or dApp on any chain wanting to check a wallet's credit reliability calls an L2 function that computes the *cosine similarity* between $\mathbf{v}_W$ and the threshold vector $\mathbf{v}_T$. If the result is above some acceptance threshold (e.g. 0.95), the wallet is deemed creditworthy. Alternatively, the lender can compare $\mathbf{v}_W$ to the credit vector of another wallet $W'$ that they previously transacted with and trust. The system returns

$$\text{cosine\_sim}(\mathbf{v}_W, \mathbf{v}_{W'}).$$

### 10.4. Verification Fee and Cost Recoup

When a cross-chain user requests official verification of $W$'s credit status on the L2, the protocol checks the *accumulated cost* of all updates to $W$'s score since the last verification. The user pays a verification fee proportional to that cost, which is burnt. This ensures the cost of repeated computations is eventually borne by those who directly benefit from the credit data.

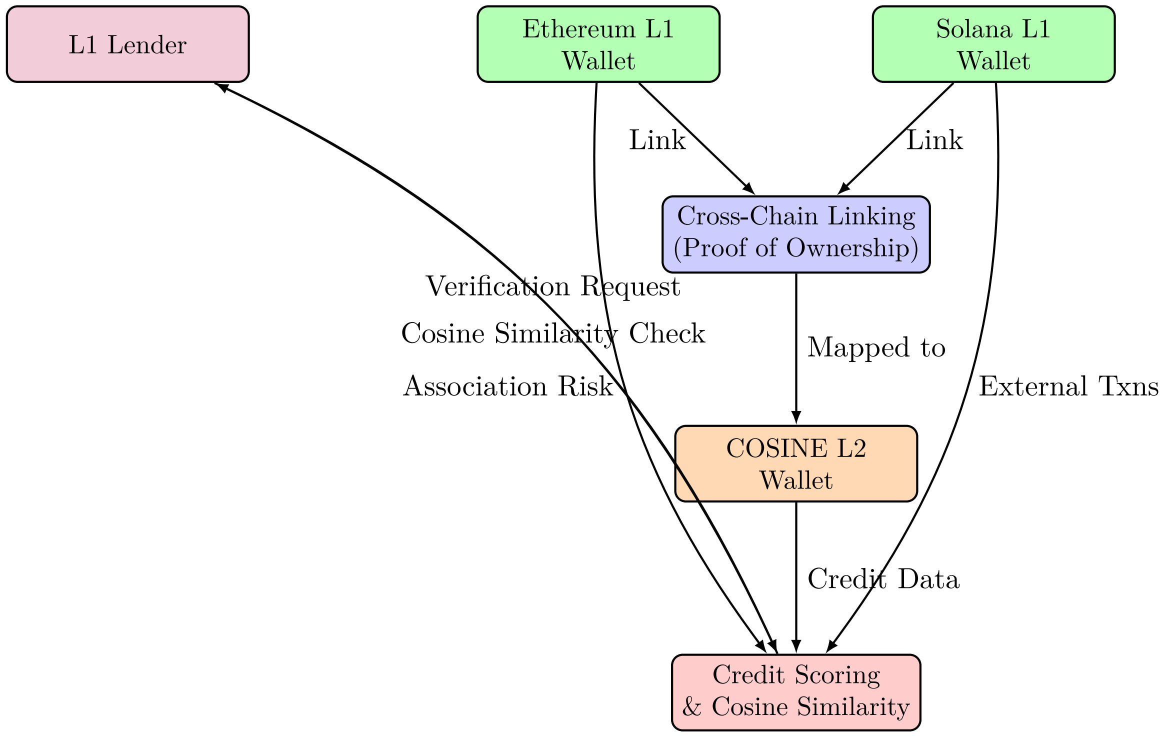

Figure 5: Cross-Chain Wallet Verification and Associated Credit Scoring.

## 11. End-to-End Workflows

This section provides several detailed scenarios illustrating how COSINE operates in practice, focusing on the *sequential voting*, *dynamic scaling*, *association penalty*, *cosine similarity*, and *cross-chain* aspects.

### 11.1. Workflow 1: Wallet Creation & Baseline Score

1. **New L2 Wallet:** A user creates a fresh wallet $W$ on the COSINE L2. The protocol initializes $C_W = C_{\text{default}}$, e.g. 0.
2. **Sparse Early Data:** Rolling statistics $\mu_V(t)$ or $\sigma_V(t)$ are initially seeded with default values. Over time, each new vote or penalty event refines these EMAs.
3. **Potential External Linking:** If the user signs with an L1 address, that address is associated to $W$, possibly affecting $C_W$ if the L1 address has suspicious history.

### 11.2. Workflow 2: Negative Vote & Score Drop

1. **Suspicion:** Some on-chain watchers suspect that wallet $W_{\text{bad}}$ performed malicious behavior (scam or exploit).
2. **User $U$ Votes:** A known user $U$ with reputation $R_U = 0.85$ casts a negative vote ($v_U = -1$).
3. **Deviation Calculation:**
$$\Delta V = (0.85 \times (-1)) - \mu_V(t).$$
Suppose the protocol sees $\Delta V < -\lambda\, \sigma_V(t)$.
4. **Score Update:**
$$\Delta_{\text{vote}} = -\, K_{\text{neg}}(t) \left(\frac{|\Delta V|}{\sigma_V(t)}\right)^{\gamma},$$
so
$$C_{W_{\text{bad}}}^{(\text{new})} = C_{W_{\text{bad}}}^{(\text{old})} + \Delta_{\text{vote}}.$$
5. **Compute Observed and Expected Impacts:** The system computes the observed and expected impacts for the negative vote.
   - The observed impact is the magnitude of the actual shift applied:
$$\text{ObservedImpact}(t) = |\Delta_{\text{vote}}^{\text{obs}}(i)| = \left| -K_{\text{neg}}(t) \cdot \left(\frac{|\Delta V(i)|}{\sigma_V(t)}\right)^{\gamma} \right|$$
   - The expected impact is based on the prior scaling factor:
$$\text{ExpectedImpact}(t) = |\Delta_{\text{vote}}^{\text{exp}}(t)| = \left| K_{\text{neg}}(t-1) \cdot \left(\frac{|\Delta V(i)|}{\sigma_V(t)}\right)^{\gamma} \right|$$

   These values are used in the Kalman filter update for $K_{\text{neg}}(t)$ in the next step.

6. **Scaling Factor Feedback Loop:** The Kalman filter updates $K_{\text{neg}}(t+1)$ based on the difference between observed and expected impacts, as detailed in Section 8.

As more negative votes accumulate, $W_{\text{bad}}$'s score can drop well below typical lending thresholds.

### 11.3. Workflow 3: Association Risk & Partial Penalty

1. **Trigger:** $W_{\text{bad}}$ is flagged malicious. The protocol initiates a random window $[t_{\text{flag}}, t_{\text{flag}} + T_{\text{limit}}]$ and a random hop limit $H_{\text{limit}}$.
2. **Association Analysis:** The system checks all addresses that receive funds from $W_{\text{bad}}$ (directly or indirectly up to $H_{\text{limit}}$ hops) within the time window. Let's consider a wallet $W_{\text{assoc}}$ that appears in this set.
3. **Transaction Weight Computation:** For each relevant transaction $i$,
$$w_i = \log(1 + A_i)\,[1 + \beta(n_{\text{cluster}} - 1)]\; e^{-\delta(t_{\text{current}} - t_i)}.$$
4. **Aggregate Risk:**
$$R_{\text{assoc}}(W_{\text{assoc}}) = \sum_{i \in \mathcal{T}} w_i.$$
5. **Rolling Mean & Std Dev:** The system has $\mu_R(t)$ and $\sigma_R(t)$. If
$$(R_{\text{assoc}}(W_{\text{assoc}}) - \mu_R(t)) > \lambda_{\text{assoc}}\,\sigma_R(t),$$
then
$$\Delta_{\text{assoc}} = -\,K_{\text{assoc}}(t)\left(\frac{R_{\text{assoc}}(W_{\text{assoc}}) - \mu_R(t)}{\sigma_R(t)}\right)^{\gamma}.$$
6. **Score Penalty:**
$$C^{(\text{new})}_{W_{\text{assoc}}} = C^{(\text{old})}_{W_{\text{assoc}}} + \Delta_{\text{assoc}}.$$
7. **Voting Restrictions:** $W_{\text{assoc}}$ is not allowed to cast a vote to rehabilitate $W_{\text{bad}}$.

### 11.4. Workflow 4: Rehabilitation via Positive Votes

1. **Initiation:** Suppose $W_{\text{bad}}$ tries to rectify the harm or proves innocence. Some high-reputation wallets cast positive votes ($v_U = +1$).
2. **Deviation Calculation:**
$$\Delta V = (R_U \cdot (+1)) - \mu_V(t)$$
If $\Delta V > +\lambda\,\sigma_V(t)$, then a positive shift occurs:
$$\Delta_{\text{vote}} = +\,K_{\text{pos}}(t)\left(\frac{|\Delta V|}{\sigma_V(t)}\right)^{\gamma}$$
Note: $K_{\text{pos}}(t)$ is dynamically updated using a Kalman filter based on observed versus expected vote impacts (see Section 8).

3. **Score Increase:**
$$C_{W_{\text{bad}}}^{(\text{new})} \;=\; C_{W_{\text{bad}}}^{(\text{old})} + \Delta_{\text{vote}}$$
Over many such votes, the wallet might be brought back above typical lending thresholds.

4. **Optionally: Additional Rehab Mechanism:** If the community uses a dedicated "rehab" vote wave, then
$$\Delta_{\text{rehab}} = +\,K_{\text{rehab}}(t)\left(\frac{(V_{\text{rehab}} - \mu_{R,V}(t))_{+}}{\sigma_{R,V}(t)}\right)^{\gamma}$$
where $K_{\text{rehab}}(t)$ is also updated via a Kalman filter (see Section 8). The final credit score shift is applied.

### 11.5. Workflow 5: Cross-Chain Borrowing & Cosine Similarity Check

The process of borrowing and lending across different blockchains is facilitated by the COSINE protocol through a series of interactions between L1 addresses and L2 COSINE wallets. This section outlines the steps for cross-chain borrowing and lending using COSINE's similarity check mechanism.

#### 11.5.1. Borrowing and Lending with Attached Wallets

Once an L1 address $W^{(1)}$ is confirmed to belong to user $U$'s COSINE wallet $W^{(2)}$, that user can borrow or lend assets in two distinct ways:

1. **On COSINE (L2) Directly:**
   - The user borrows or lends in COSINE tokens (or any whitelisted tokens bridging to L2).
   - All transactions occur on the COSINE network, and the protocol automatically records on-chain flows.
   - Because the transaction is native to L2, the protocol trivially knows that user $B$ transacted with user $A$.

2. **On the Underlying L1 Blockchain:**
   - The user or lender chooses to transfer, for example, Bitcoin, ETH, or SOL directly on L1.
   - COSINE only sees that user $A$ (COSINE ID: $W_A^{(2)}$) claims to be receiving or sending funds at L1 address $W_A^{(1)}$.
   - The protocol requests minimal feedback from the L1 lender or counterparty (user $B$) afterward, as described below.

In both cases, COSINE aims to maintain an accurate credit score for user $A$, factoring in the success or failure of any repayment.

### 11.5.2. L1 $\rightarrow$ L2 Verification for Lenders or User-to-User Transactions

Suppose user $B$ is a lender or a DeFi platform on L1 and wants to evaluate user $A$'s credibility. The recommended workflow is as follows:

1. **User $A$ Submits Identity:**
   - If user $A$ wishes to borrow on L1, they provide either their L1 address $W_A^{(1)}$ *or* the corresponding COSINE wallet ID $W_A^{(2)}$.
   - A user interface or API call to COSINE's "CheckScore" function returns a zero-knowledge proof or aggregated measure that $\text{score}(W_A^{(2)}) \geq T$, for some threshold $T$.
2. **Look-Up of L2 Wallet from L1 Address:**
   - If user $A$ provides the L1 address only, the lending platform queries COSINE's association mapping $\mathcal{M}$ to see the mapped L2 wallet $W_A^{(2)}$. This is a direct key–value lookup.
   - The "CheckScore" function confirms whether the L2 wallet's reputation meets the requested threshold.
3. **Decision on Lending:**
   - If the L2 credit score is acceptable, user $B$ proceeds with the loan on L1. Otherwise, they may reject or require more collateral.

### 11.5.3. Post-Transaction Feedback from L1 or L2 Counterparties

1. **Feedback Requirement:** COSINE enforces a rule that *before* user $B$ can do any subsequent verification or new loan checks on the L2 network, they must answer a few questions:
   - If the transaction (loan) happens on L1—i.e., user $B$ actually sends assets to user $A$'s L1 address
   - COSINE enforces a rule that *before* user $B$ can do any subsequent verification or new loan checks on the L2 network, they must answer the question: "Did you transact with user $A$?"
   - If yes, they must also cast a minimal "Trust or No-Trust" vote on user $A$.
   - If the transaction (loan) happens on L2—i.e., user $B$ actually sends assets to user $A$'s L2 address. Since the Cosine Network already knows a transaction happened.
   - User $B$ just cast a minimal "Trust or No-Trust" vote on user $A$
2. **Impact on $A$'s Credit Score:**
   - If user $B$ states they transacted and had no issues, a small positive shift to $A$'s embedding can be applied (subject to standard aggregator checks in COSINE).
   - If user $B$ states they transacted and "do not trust" user $A$, this triggers a negative shift or at least flags for further investigation/voting.

### 11.5.4. Verification Function and Cosine Similarity

For cross-chain lending and borrowing, the following verification function is used:

- **Verification Function:** The lender on Ethereum calls ComputeCosineSimilarity$(W_A^{(2)}, T)$ on the COSINE L2, paying the verification fee if needed.
- **Normalization:** The protocol normalizes the wallet's credit score:

$$\hat{C}_{W_A^{(2)}} = \frac{C_{W_A^{(2)}} - \mu_C}{\sigma_C}.$$

  The threshold is similarly normalized: $\hat{T}$.

- **Vector Embedding & Cosine Similarity:** The protocol embeds the wallet's score and threshold as vectors:

$$\mathbf{v}_W = \begin{pmatrix} \hat{C}_{W_A^{(2)}} \\ 1 \end{pmatrix}, \quad \mathbf{v}_T = \begin{pmatrix} \hat{T} \\ 1 \end{pmatrix}.$$

  Then, the cosine similarity is calculated as:

$$\text{similarity} = \frac{\mathbf{v}_W \cdot \mathbf{v}_T}{\|\mathbf{v}_W\| \, \|\mathbf{v}_T\|}.$$

- **Decision:** If similarity $\geq$ thresholdSim (e.g., 0.95), the lender proceeds with the loan. Otherwise, the borrower may be seen as risky.

### 11.5.5. Efficiency Considerations

- The association mapping $\mathcal{M}$ is updated off-chain by a random subset of validators and committed periodically. This ensures that queries are quick (single or few lookups) while still being fully decentralized in maintenance.

### 11.6. Workflow 6: Malicious Re-Linking Attempt

1. **Blacklisted L1 Address:** Suppose $W^{(1)}_{\text{bad}}$ is a malicious address on Ethereum. Its linked L2 wallet $W^{(2)}_{\text{bad}}$ is also penalized.
2. **Creating Fresh Addresses:** The attacker transfers funds to a new L1 address $W^{(1)}_{\text{new}}$, hoping to link it to a newly created L2 wallet $W^{(2)}_{\text{new}}$ and avoid the old penalties.
3. **Association Risk at Linking:** Once the attacker tries to finalize the link, the protocol performs multi-hop tracing. It finds that $W^{(1)}_{\text{new}}$ recently received funds from $W^{(1)}_{\text{bad}}$ within the random time/hop constraints.
4. **Partial Penalty:**
$$C^{(\text{new})}_{W^{(2)}_{\text{new}}} = C^{(\text{old})}_{W^{(2)}_{\text{new}}} + \Delta_{\text{assoc}},$$
effectively penalizing the new wallet.
5. **Ongoing Monitoring:** If $W^{(2)}_{\text{new}}$ continues suspicious activity, negative votes or further association triggers can push its score even lower.

#### 11.6.1. Workflow 7: Linking the Same L1 Address to Multiple COSINE Wallets

COSINE prohibits the same L1 address $W^{(1)}$ from being linked to more than one L2 address $W^{(2)}$. The mapping $\mathcal{M}$ ensures:

- **If** user $A_{\text{bad}}$ with a blacklisted COSINE wallet tries to create a new L2 wallet and re-link the same L1 address, the protocol rejects it.
- The VRF-selected validators performing the link request will detect that $\mathcal{M}[\,W^{(1)}\,]$ is already set, thus denying the attempt.

These scenarios demonstrate how COSINE's self-tuning, sequential approach manages negative votes, partial blacklisting, rehabilitations, cross-chain interactions, and final verification steps.

## 12. Transaction Fee Mechanism

COSINE sustains its validator set and ensures cost coverage by employing a **fee-based recoup** design. Each update event draws from a network reward pool, and each *verification* transaction repays the system by burning the accumulated cost.

### 12.1. Per-Event Payments to Validators

Whenever a wallet $W$ undergoes a credit score update event (due to a vote, association penalty, or rehab shift), the protocol pays a reward $R_{\text{total}}^{(i)}$ to the validators from its network pool. Over multiple updates, the total cost for $W$ accumulates:

$$C_{\text{acc}}(W) = \sum_{i=1}^{n} R_{\text{total}}^{(i)}.$$

### 12.2. Verification Fee & Burn

When a user or DAO calls VerifyScore($W$), the protocol checks $C_{\text{acc}}(W)$. The caller must pay:

$$F_{\text{verif}}(W) = \kappa \times C_{\text{acc}}(W),$$

where $\kappa \geq 1$. That amount is then burned. Finally, $C_{\text{acc}}(W)$ is reset to 0. This ensures that repeated or frequent updates are not "free," but eventually recouped at the point of verification by the beneficiaries who require the credit score.

### 12.3. Cosine Similarity Request

If a user wants the protocol to compute cosine_sim$(\mathbf{v}_W, \mathbf{v}_T)$ or cosine_sim$(\mathbf{v}_W, \mathbf{v}_{W'})$ with official on-chain proof, they must pay the same verification fee. This covers the cost of prior updates that shaped $C_W$. A user can still *read* the raw $C_W$ by running their own node or explorer, but official on-chain verification is locked behind the fee mechanism, ensuring cost coverage.

### 12.4. Rationale

- **Cost Coverage:** Over many updates, the reward pool pays validators. Ultimately, the user or DAO that *benefits* from the final credit verification reimburses those costs.
- **Stability:** Because the verification fee is burnt, circulating token supply remains stable with network usage as validators are paid.
- **Spam Deterrence:** Excessive updates for a single wallet make its subsequent verification more expensive, discouraging trivial spamming of the credit score.

## 12.5. Transaction Fees for Regular Token Transfers

In addition to fees for credit score verifications, the COSINE protocol implements a distinct fee mechanism for regular send and receive transactions of COSINE tokens directly on the Layer-2 (L2) blockchain. These transactions, which involve transferring tokens between wallets, are fundamental to the token's utility as a medium of exchange within the COSINE ecosystem. This fee structure ensures that the computational and storage costs of processing such transactions are covered while incentivizing validators to maintain network operations, all while aligning with the protocol's validator-driven consensus (Section 4) and decentralized storage framework (Section 5).

**Fee Structure** Each token transfer transaction requires the sender to pay a fee in COSINE tokens, comprising two components:

- **Base Fee:** A minimum fee set by the protocol to cover the computational cost of transaction validation and the storage cost of updating the Wallet State Trie (WST). This fee is dynamically adjusted based on network usage (e.g., transaction volume over a trailing period) to prevent spam and ensure resource efficiency. For example, the base fee might be initialized at 0.002 COSINE tokens, slightly above the estimated marginal cost of processing and storage (e.g., 0.001 COSINE tokens).
- **Optional Tip:** An additional, user-specified amount that can be included to prioritize the transaction during periods of high demand. While optional, the tip incentivizes validators to include the transaction in the next block when block space is contested, enhancing user flexibility.

The total fee for a transaction $i$ is thus:

$$F_i = \text{Base Fee}_i + \text{Tip}_i.$$

**Fee Collection and Distribution** The process for collecting and distributing fees leverages the existing validator architecture:

1. **Block Production:** A subset of validators, $V_{\text{selected}}$, is chosen via the Verifiable Random Function (VRF) mechanism (Section 4.2) to propose and validate each block. Blocks are produced periodically (e.g., every 5 minutes) or triggered by sufficient transaction volume, as outlined in Section 5.
2. **Transaction Inclusion:** The block proposer assembles a block from the transaction pool, prioritizing transactions with higher total fees ($F_i$) if block space is limited due to congestion. In typical conditions, COSINE's high-throughput L2 design ensures all transactions meeting the base fee are included.
3. **Fee Pooling:** Upon block confirmation, the total fees from all $n$ transactions in the block are pooled:
$$\text{Pooled Fees} = \sum_{i=1}^{n} F_i.$$

4. **Distribution to Validators:** The pooled fees are distributed equally among the $m = |V_{\text{selected}}|$ validators in the subset:

$$R_{v,i} = \frac{\text{Pooled Fees}}{m},$$

where $R_{v,i}$ is the reward paid to each validator $v \in V_{\text{selected}}$. This equal distribution is justified since the VRF selection process already accounts for validators' stake ($S_v$) and performance scores ($P_v$) through the weight $W_v = S_v^\alpha \times (1 + \beta P_v)$, ensuring that higher-performing validators are selected more frequently and thus earn more over time.

**Integration with Decentralized Storage** Regular token transfers directly update the COSINE L2 ledger, specifically the Wallet State Trie (WST) within the decentralized storage system (Section 5). Each transaction involves:

- Verifying the sender's signature.
- Checking the sender's token_balance in the WST to ensure sufficient funds (amount plus fee).
- Updating the sender's and receiver's token_balance fields in the WST (subtracting the amount and fee from the sender, adding the amount to the receiver).
- Recording the transaction in the block's transaction list, formatted as $\{\text{sender}, \text{receiver}, \text{amount}, \text{timestamp}\}$.

These updates result in a new Merkle root (WST_root) committed to the block header, ensuring tamper-proof storage. The base fee is calibrated to exceed the computational and storage costs of these operations, which are relatively constant per transaction, unlike the variable costs of credit score updates.

**Rationale**

- **Cost Coverage:** The base fee ensures that the marginal costs of processing and storing each transaction are met, maintaining the economic viability of the L2 network.
- **Validator Incentives:** Direct fee distribution to validators complements the network reward pool payments for credit score updates (Section 4.5), providing a self-sustaining incentive for block production.
- **Separation from Credit Score Mechanisms:** Keeping transaction fees distinct from verification fees (Section 12) and network pool rewards preserves funding for the protocol's credit scoring and fraud mitigation functions, while ensuring regular token transfers operate independently.
- **User Flexibility:** The optional tip allows users to prioritize transactions during peak usage, enhancing the user experience in a scalable Layer-2 environment.
- **Spam Deterrence:** Transactions with fees below the base fee are rejected or delayed, preventing spam without requiring a complex gas system.

**Implementation Notes** The base fee is initially set via protocol parameters and can be adjusted through community governance or an automated algorithm (e.g., based on average transaction volume over a 24-hour window). Client software (e.g., wallets) will suggest a default fee (base fee plus a recommended tip) based on current network conditions, simplifying the process for users. This mechanism ensures that COSINE remains a high-throughput, user-friendly L2 blockchain while supporting its broader goals of trust and fraud mitigation in decentralized finance.

## 13. Tokenomics

The total COSINE token supply remains:

$$\boxed{1\,000\,000\,000 \text{ COSINE tokens}}$$

Distribution is:

- **Foundation & Core Team:** 10%
- **Advisors & Contributors:** 5%
- **Early Developer/Builder Grants:** 10%
- **Private Sale / Presale (combined):** 15%
- **Public Sale:** 20%
- **Network Rewards & Ecosystem Funds:** 40%

The 40% reserved for Network Rewards & Ecosystem Funds specifically covers $R^{(i)}_{\text{total}}$ validator payments. Over time, the recoup mechanism ensures that active users or DAOs eventually burn an equivalent or larger sum in verification fees.

**Vesting Schedule:**

- **Foundation & Core Team (10%):** Locked under a 4-year vesting schedule with a 1-year cliff; then linear monthly unlocks over the next 36 months.
- **Advisors & Contributors (5%):** Locked under a 2-year vesting schedule with a short cliff or linear monthly unlocks.
- **Early Developer/Builder Grants (10%):** Released based on milestone achievements.
- **Private Sale / Presale (15%):** 6–12 months lock, then linear unlock over an additional 6–12 months.
- **Public Sale (20%):** Unlocked immediately at the Token Generation Event (TGE).
- **Network Rewards & Ecosystem Funds (40%):** Distributed steadily over 5–8+ years.

### 13.1. Multi-Chain Strategy & Supply Invariance

COSINE tokens will be deployed on all major blockchain networks. The total supply remains fixed at 1 billion tokens. Bridging contracts lock tokens on Ethereum and mint equivalent wrapped tokens on other chains using a lock-and-mint (or burn-and-redeem) mechanism. Cross-chain arbitrage ensures that token prices converge across different ecosystems.

### 13.2. Validator Staking Requirement

To join the validator set of the COSINE network, participants are required to purchase and stake a minimum of **100,000 COSINE tokens**. This requirement ensures that validators have a significant economic commitment to the protocol, aligning their incentives with the long-term security and integrity of the network. The staked tokens serve as collateral against potential misbehavior and along with performance determine each validator's influence within the consensus mechanism.

## 14. Roadmap

### 14.1. Phase 1: Foundation & Early Launch (0–6 Months)

1. **Core Protocol & MVP:** Implement the fundamental self-tuning, single-dimensional credit scoring system with VRF-based validator selection, outlier filtering, and the cost recoup mechanism. Deploy on an Ethereum testnet.
2. **Presale & Public Sale:** Conduct private/presale rounds to raise capital, followed by a public sale (20% of tokens) on Ethereum at TGE, and list COSINE on at least one DEX.
3. **Developer SDKs & Community Bootstrapping:** Allocate tokens to early testers, advisors, and developers (under vesting schedules) and publish developer SDKs.

### 14.2. Phase 2: Cross-Chain Deployment & Lending Partnerships (6–12 Months)

1. **Bridging Contracts:** Deploy bridging contracts to major L1 chains (Ethereum, Solana, Binance) and establish liquidity on major DEXs.
2. **Lending Integration:** Form partnerships with DeFi lending platforms to use the COSINE credit scoring and partial penalty logic.
3. **Validator & Developer Incentives:** Begin distributing network rewards from the 40% pool to validators and developers, and release milestone-based developer grants.

### 14.3. Phase 3: Global Expansion & DAO Formation (12–24 Months)

1. **COSINE Lending DAO:** Establish a governance framework for real-world lending integration, weighting parameters (e.g. initial $\lambda, \gamma, \beta$), and advanced proposals.
2. **Open Lending Framework:** Provide an easy interface for any project to plug into the COSINE system for credit checks and partial penalty logic.
3. **CEX Listings & Broad Adoption:** Seek listings on top centralized exchanges and integrate further across DeFi ecosystems.

### 14.4. Phase 4: Advanced Scaling & Full Ecosystem (24+ Months)

1. **DAO Evolution & On-Chain Governance:** Transition to fully on-chain community management of advanced parameters, synergy with real-world credit bureaus, etc.
2. **Global Real-World Adoption:** Expand cross-industry partnerships to bridge traditional finance institutions, banks and online lending platforms with COSINE scoring.
3. **Ongoing Optimizations & Research:** Continue refining the self-tuning approach, analyzing advanced outlier detection, or exploring new cross-chain frameworks.

## 15. Conclusion

COSINE offers a transformative approach to decentralized credit scoring and fraud mitigation within the rapidly evolving DeFi ecosystem. By leveraging a self-tuning, real-time scoring mechanism, COSINE enables adaptive and transparent credit assessments, providing a trust-minimized environment for decentralized lending and financial interactions. The system's integration of reputation-weighted voting, dynamic scaling factors, and robust fraud detection mechanisms—including randomized time windows and multi-hop association analysis—ensures that the protocol is both resilient to evolving malicious strategies and efficient in updating creditworthiness in response to on-chain behavior.

The use of cosine similarity for credit verification not only offers a more stable and intuitive measure of trust but also enhances interoperability across multiple blockchains, empowering DeFi participants to engage in cross-chain transactions with confidence. Additionally, the protocol's cost-recovery mechanism ensures sustainability, aligning incentives between users, validators, and the broader ecosystem.

COSINE represents a critical step forward in the pursuit of a decentralized financial system that operates on trust derived from verifiable, collective consensus rather than centralized authorities. As the DeFi space continues to grow, COSINE's framework for real-time credit scoring and fraud mitigation sets a new standard for how decentralized applications can scale, innovate, and maintain integrity while protecting against fraudulent activities. Ultimately, COSINE paves the way for a more secure, efficient, and inclusive DeFi ecosystem, where credit and trust are governed by the collective actions of its participants.

## 16. References

### References

[1] **Wired.** “The Pig Butchering Invasion Has Begun.” Published October 22, 2024.

[2] **Reuters.** “Losses from crypto scams grew 45% in 2023, FBI says.” Published September 9, 2024.

[3] **The Atlantic.** “When the Bitcoin Scammers Came for Me.” Published September 5, 2024.

[4] **Wikipedia.** “Cryptocurrency and Crime.” Published October 2024.

[5] **Bankless.** “The Ultimate Guide to Undercollateralized Lending in DeFi.” Published November 16, 2022.

[6] **Switcheo Blog.** “Meet Zero Collateral. Undercollateralized DeFi Loans on. . . .” Published 2.3 years ago.

[7] **Tally.** “Treasury Building Blocks: Undercollateralized Lending.” Published Jul 29, 2021.

[8] **Medium.** “Meet Zero Collateral. Undercollateralized DeFi Loans on. . . ” Published 5.1 years ago.

[9] **Kraken.** “Crypto Loans: What They Are and How They Work.” Published July 26, 2024.

[10] **RiskSeal.** "The Role of AI in Modern Credit Risk Management." Published December 2024.

[11] **ResearchGate.** "Implementing Dynamic Risk Scoring Models for Adaptive Fraud Prevention." Published January 2025.

[12] **Tandfonline.** "Dual-Channel Memory Networks for Fraud Detection." Published December 2024.

[13] **ACM Digital Library.** "Federated Graph Learning for Credit Card Fraud Detection." Published January 2025.

[14] **Wikipedia.** "Kalman Filter." Accessed February 19, 2025.

[15] **GeeksforGeeks.** "Overview of Kalman Filter for Self-Driving Car." Published 2.3 years ago.

[16] **arXiv.** "Probabilistic 3D Multi-Object Cooperative Tracking for Autonomous Driving via Differentiable Multi-Sensor Kalman Filter." Published 1.4 years ago.

### Contact Info

**Email :** info@iwalink.ch
**Website :** cosinelayer2.com