entropy

MDPI

Article

VeriFed: Temporally Consistent Continuous Cross-Chain Data Federation

Kun Hao [1,*], Meng Bi [1,*] and Yuliang Ma [2]

[1] School of Software, Shenyang University of Technology, Shenyang 110870, China
[2] College of Information Science and Engineering, Northeastern University, Shenyang 110819, China; mayuliang@mail.neu.edu.cn
* Correspondence: haokun@sut.edu.cn (K.H.); bim@sut.edu.cn (M.B.)

Abstract

Cross-chain analytics increasingly demand *continuous* joins across ledgers with asynchronous state evolution. Existing solutions, however, typically assume static snapshots or neglect temporal alignment, yielding semantically inconsistent results when epochs drift. This paper introduces VeriFed, a system for temporally consistent continuous cross-chain joins. We formalize the problem of snapshot-aligned continuous joins, design a Unified Adapter Layer (UAL) to align finalized snapshots across heterogeneous protocols, and develop *incremental verification* that composes per-chain proofs into a global summary via the Epoch Attestation Mesh (EAM) and the Delta-Linked Proof Forest (DLPF). To sustain high-throughput execution, VeriFed further adopts an incremental multi-objective optimizer that balances latency and monetary cost. Experiments on Ethereum transaction data with a simulated wide-area network (WAN) demonstrate that VeriFed achieves sub-second per-epoch latency (approx. 38 ms) and reduces verification overhead by orders of magnitude compared to state-of-the-art baselines, while effectively detecting tampering with zero false positives. These results confirm consistent efficiency and verifiability under continuous updates.

Keywords: blockchain; cross-chain query; join query processing; verifiable query; distributed systems

1. Introduction

The blockchain ecosystem has transitioned from isolated networks into a complex archipelago of heterogeneous distributed ledgers [1], including Layer-1 blockchains (e.g., Ethereum [2], Solana [3]) and Layer-2 scaling solutions (e.g., Optimism [4], Arbitrum [5]). As decentralized finance (DeFi) [6] and supply chain applications [7] expand across these boundaries, there is a critical demand for *cross-chain analytics* that can correlate data from multiple independent sources [8]. For instance, detecting money laundering often requires joining transaction graphs across Bitcoin and Ethereum; similarly, cross-border supply chains track assets moving between permissioned Hyperledger Fabric [9] instances and public chains. Unlike simple asset transfers handled by bridges, these applications require executing complex relational queries—specifically *continuous joins*—over the evolving states of multiple ledgers.

Realizing such analytics, however, faces a fundamental hurdle: temporal inconsistency caused by the inherent asynchrony of distributed ledgers. From an information-theoretic perspective, this temporal misalignment can be viewed as a form of System Entropy,

Check for updates

Academic Editor: Boris Ryabko

Received: 13 March 2026
Revised: 13 April 2026
Accepted: 15 April 2026
Published: 21 April 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

where the disorder in the global state increases as the constituent ledgers drift apart asynchronously. Without a mechanism to minimize this entropy, the joined information becomes unreliable. Unlike centralized databases synchronized by a global clock, blockchains produce blocks at varying, often unpredictable rates. Naively querying the "latest" block from each chain yields a *temporally skewed* view—a phantom global state that never existed. For example, in a cross-chain arbitrage bot, joining a price oracle update from Chain A (timestamp t) with a liquidity pool state from Chain B (timestamp $t - \delta$) can lead to substantial financial losses if δ exceeds the market volatility threshold. Furthermore, as these queries are often continuous (e.g., monitoring streams of events), the system must maintain this temporal alignment epoch-by-epoch while coping with the dynamic drift between chain clocks.

State-of-the-art solutions fail to reconcile these requirements. On-chain interoperability protocols (e.g., Polkadot's XCMP [10], Cosmos IBC [11]) prioritize asset atomicity over analytical consistency, making them prohibitively expensive and slow for data-intensive relational joins. Off-chain indexers (e.g., The Graph [12], Dune Analytics) provide SQL-like interfaces but rely on centralized or federated indexing servers, sacrificing the trustless verification guarantees inherent to blockchains. Meanwhile, Verifiable Query Processing (VQP) schemes designed for single chains [13–15] induce prohibitive cryptographic overhead when applied to multi-chain streams, as clients must verify independent proofs from every source ledger for each update, leading to a bottleneck in computation and bandwidth.

In this paper, we introduce VeriFed, a system designed to enable verifiable, temporally consistent, and continuous cross-chain data federation. We address three primary technical challenges:

1. **Snapshot Alignment under Heterogeneity:** how to define and efficiently capture a coherent global snapshot across asynchronous chains with different finality rules (e.g., probabilistic PoW vs. deterministic PBFT).
2. **Incremental Verification Scalability:** how to minimize the verification cost for high-frequency continuous streams, avoiding the redundant re-verification of static historical data.
3. **Dynamic Multi-Objective Optimization:** how to optimize the execution plan (e.g., join order, data transfer direction) dynamically as data distributions and network costs (gas fees) fluctuate over time.

VeriFed addresses these challenges through a layered architecture. First, we define a *skew-bounded* consistency model, where the system ensures that all source blocks in a query epoch fall within a user-defined time window Δ. To achieve efficient verification, we propose the Epoch Attestation Mesh (EAM) and the Delta-Linked Proof Forest (DLPF). The EAM leverages a committee of verifiers to inspect query results and generate a consolidated signature, while the DLPF structures cryptographic proofs incrementally—clients only need to verify the *delta* (changes) between consecutive epochs rather than the full state, significantly reducing overhead. Finally, we design an incremental multi-objective optimizer that continuously monitors system metrics and adjusts execution plans to balance result latency and monetary costs.

The specific contributions of this paper are as follows:

- **Formalization of Snapshot-Aligned Continuous Joins:** We formally define the problem of continuous cross-chain joins with strict temporal skew bounds (Δ) and model the alignment fidelity from an information-theoretic perspective.
- **VeriFed System Architecture:** We present VeriFed, a comprehensive framework that integrates a Unified Adapter Layer (UAL) for snapshot alignment and an execution engine for continuous query processing.

https://doi.org/10.3390/e28040478

- **Incremental Verification Protocol:** We design the EAM and DLPF mechanisms, which allow lightweight clients to verify continuous result streams with minimal computational cost by exploiting the temporal locality of blockchain state changes.
- **Multi-Objective Incremental Optimization:** We propose an optimization algorithm that dynamically adapts query execution plans in response to real-time changes in data selectivity and gas prices.
- **Extensive Evaluation:** We implement a prototype of VeriFed and evaluate its performance using real-world Ethereum transaction data simulated across a Wide Area Network (WAN). The results demonstrate that VeriFed achieves sub-second latency (approx. 38 ms) and over 60,000 TPS, significantly outperforming existing single-chain verifiable query systems while maintaining strict verifiability and temporal consistency.

The remainder of the paper is organized as follows. Section 2 presents the problem definition and threat model. Section 3 provides a system overview. Sections 4–6 describe snapshot-aligned continuous joins, incremental verification, and incremental multi-objective optimization. Section 7 presents the security model and proofs. Section 8 reports the experimental results. Section 9 reviews related work. Section 10 concludes the paper.

2. Problem Definition and Threat Model

In this section, we formalize the system model, strictly define the problem of verifiable continuous cross-chain joins, and outline the specific threats targeting stream processing and verification.

2.1. System Model

We consider a cross-chain ecosystem $\mathcal{C} = \{C_1, \ldots, C_n\}$. Each blockchain C_i is an append-only sequence of blocks $b_{i,h}$ finalizing at height h.

- **State Transition Stream:** The state $S_{i,h}$ is derived from $S_{i,h-1}$ by applying the transaction set in $b_{i,h}$. We model each blockchain as a sequence of state deltas $\delta_{i,h} = S_{i,h} \setminus S_{i,h-1}$, representing the incremental updates (inserted/deleted tuples) at each height.
- **Finality Oracle:** A function $\mathcal{F}_i(h) \to \{\texttt{True}, \texttt{False}\}$ indicates if block $b_{i,h}$ is finalized. VeriFed only processes h where $\mathcal{F}_i(h) = \texttt{True}$.

2.2. Problem Formulation: Verifiable Continuous Join

Unlike one-time queries, a *continuous* join must maintain consistency over an evolving multi-chain state. We define this as a stream transformation problem.

Definition 1 (Δ-Aligned Epoch). *A discrete time unit e is associated with a snapshot vector $H^{(e)} = \langle h_1^{(e)}, \ldots, h_n^{(e)} \rangle$. The epoch is Δ-Valid if the following conditions are satisfied: 1. All $h_i^{(e)}$ are finalized. 2. Timestamp skew satisfies $\max_{i,j} |t_{i,h_i^{(e)}} - t_{j,h_j^{(e)}}| \leq \Delta$. 3. Monotonicity: $H^{(e)} \geq H^{(e-1)}$ (component-wise) to ensure stream progress.*

Definition 2 (Information Fidelity of Alignment). *Let S^{τ}_{ideal} be the hypothetical perfectly synchronized global state at wall-clock time τ. The actual aligned state $S^{(e)}_{aligned}$ derived from $H^{(e)}$ deviates from S^{τ}_{ideal} due to the latency $\Delta_i = |\tau - t_{i,h_i}|$. We define the* Information Fidelity *of the alignment as the inverse of the aggregate temporal uncertainty:*

$$F(H^{(e)}) = \frac{1}{1 + \sum_{i=1}^{n} \omega_i \cdot \Delta_i} \tag{1}$$

where ω_i is a weight factor representing the volatility or importance of chain C_i. Maximizing $F(H^{(e)})$ minimizes the temporal entropy *(uncertainty) of the cross-chain snapshot, ensuring that the joined result reflects a coherent state of reality.*

Definition 3 (Verifiable Continuous Result Stream)**.** *Given a query Q, the system produces a stream of authenticated tuples $\mathcal{S} = \langle (R^{(1)}, \Pi^{(1)}), (R^{(2)}, \Pi^{(2)}), \dots \rangle$.*

- *$R^{(e)}$ is the join result $Q(H^{(e)})$ computed over the states at epoch e.*
- *$\Pi^{(e)}$ is a cryptographic proof certifying that $R^{(e)}$ is the **exact** result of executing Q on $H^{(e)}$ and that $H^{(e)}$ satisfies the Δ-constraint.*

Figure 1 illustrates this model, where the system must continuously find an aligned cut (dashed line) across two chains that respects the Δ constraint.

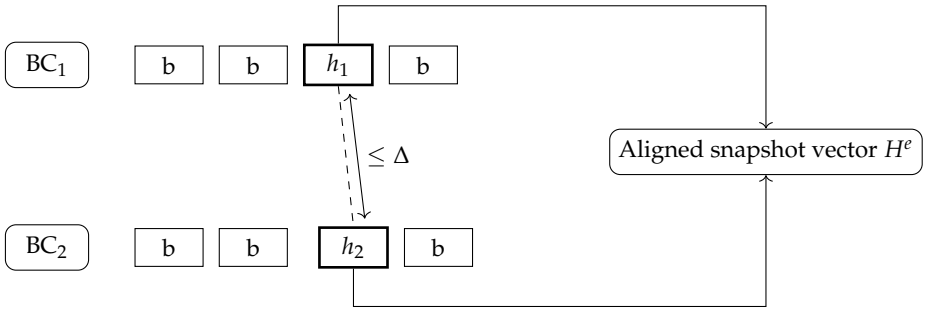


Figure 1. Threat model with an aligned snapshot vector and bounded skew across BC_1 and BC_2.

The problem requires efficiently computing $\mathcal{S}$ such that the verification cost $|\Pi^{(e)}|$ and client-side checking time are minimized (ideally $O(1)$ or $O(\log |\Delta S|)$), independent of the total state size.

2.3. Threat Model

We assume the VeriFed Service Provider (SP) is untrusted and may behave maliciously to manipulate query results. Our trust model is hybrid and explicit: (i) each underlying blockchain is trusted only up to its native security assumptions (finality threshold, honest-majority model, and canonical-fork rule), while (ii) the cross-chain attestation layer (EAM) is trusted under a BFT assumption with fewer than one-third Byzantine validators.

- **Result Tampering:** The SP alters $R^{(e)}$ (e.g., modifying prices in a DeFi join) or fabricates fake events.
- **Stream Manipulation:**
  - *Epoch Withholding:* The SP selectively drops epochs to hide specific market events.
  - *Replay Attack:* The SP presents an old valid result $(R^{(e-k)}, \Pi^{(e-k)})$ as the current state.
  - *Desynchronization:* The SP uses unaligned snapshots (violating Δ) to exploit arbitrage opportunities (e.g., front-running).
- **Collusion Risk:** The SP may collude with a subset of the verification committee (EAM). We assume the honest majority assumption holds for the underlying chains, but the off-chain EAM requires a BFT threshold $t < |M|/3$ for safety.
- **Fork/Reorganization Risk:** If a chain experiences a reorganization before finality, the corresponding header is excluded from alignment. VeriFed only accepts headers that pass the chain-specific Finality Gadget and re-checks finality before EAM signing.

2.4. Design Goals

- **Stream Integrity and Completeness:** Users must be able to verify that the result stream is unbroken and every $R^{(e)}$ contains all valid tuples.

- **Temporal Correctness:** Proofs must cryptographically bind the result to a specific Δ-aligned vector $H^{(e)}$, preventing skew attacks.
- **Incremental Efficiency:** The system must minimize recomputation. If states change slightly ($\delta_{i,h}$ is small), the cost to update $R^{(e)}$ and generate $\Pi^{(e)}$ should be proportional to $|\delta|$, not $|S|$.

3. Overview of VeriFed

VeriFed operates as a *verifiable middleware* positioned between heterogeneous blockchains and user applications. Its primary objective is to facilitate snapshot-aligned continuous joins—a class of queries demanding strict temporal consistency across independent distributed ledgers. As illustrated in Figure 2, the system adopts a layered architecture to decouple consensus abstraction, query execution, and result verification.

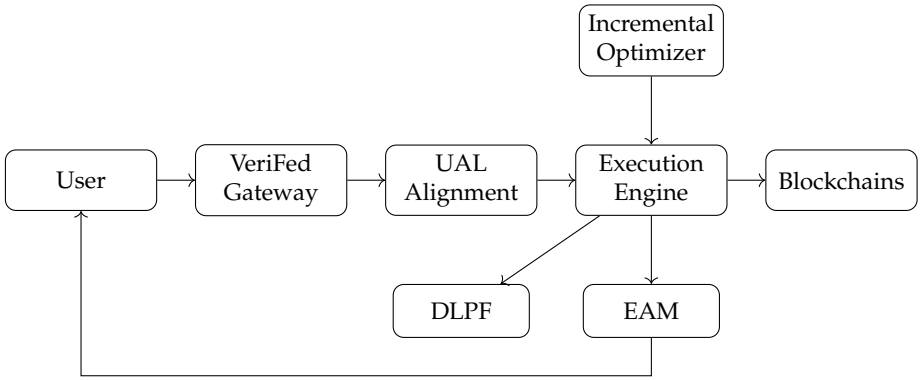


Figure 2. VeriFed architecture with UAL alignment, EAM and DLPF verification, and incremental optimization.

3.1. Architectural Layers

VeriFed comprises three vertically integrated layers, each addressing a specific challenge in cross-chain data management:

- **Unified Adapter Layer (UAL):** This layer abstracts *heterogeneity*. It connects to diverse Layer-1/Layer-2 chains (e.g., Ethereum, Solana, Arbitrum) and normalizes their block headers and Merkle proofs into a standard format. Crucially, it houses the Snapshot Alignment Scheduler, which continuously monitors finality signals to construct Δ-aligned snapshot vectors $H^{(e)}$, masking the asynchronous nature of underlying chains from the execution engine.
- **Continuous Execution Engine:** This layer drives *performance*. Unlike traditional one-shot query engines, it employs an Incremental Optimizer to generate long-running execution plans. It retrieves state deltas $\delta_{i,h}$ rather than full snapshots, updating the join result $R^{(e)}$ incrementally. This design minimizes data transfer and computation, ensuring low latency for real-time applications.
- **Verification Mesh (EAM and DLPF):** This layer anchors *trust*. The Epoch Attestation Mesh (EAM) is a decentralized committee that acts as a "verifiable witness." It aggregates per-chain cryptographic proofs into a single Global Digest D^e for each epoch. The Delta-Linked Proof Forest (DLPF) structurally organizes these proofs to support efficient historical verification, allowing clients to validate the entire result stream with constant-sized checkpoints.

3.2. Lifecycle of a Continuous Query

The lifecycle of a request in VeriFed follows a strict *Epoch-based* processing model, ensuring that every output tuple is deterministically derived from a finalized, aligned global state.

- **Query Registration:** A user submits a continuous SQL query Q (e.g., `SELECT * FROM ETH.T1, SOL.T2 WHERE ...`) along with a temporal skew constraint Δ. The system parses Q and initializes a stream cursor.
- **Epoch Initialization (Alignment):** The UAL monitors connected chains. Once a new set of finalized blocks forming a valid Δ-aligned vector $H^{(e)}$ is available, an *Epoch* e is triggered.
- **Delta Retrieval and Execution:** The engine fetches the state deltas corresponding to the transition from $H^{(e-1)}$ to $H^{(e)}$. The Incremental Optimizer selects the optimal join order (e.g., Hash Join vs. Sort-Merge) based on current gas prices and data selectivity. The engine then computes the result update $\Delta R^{(e)}$.
- **Proof Aggregation:** The EAM validates the source proofs (e.g., Merkle branches) for the retrieved deltas. It then runs a BFT consensus protocol to sign the result summary, producing the global proof $\Pi^{(e)}$.
- **Stream Delivery:** The tuple $(R^{(e)}, \Pi^{(e)})$ is pushed to the client. The client verifies $\Pi^{(e)}$ against the previous state $\Pi^{(e-1)}$ and the blockchain headers, ensuring the stream's integrity without re-executing the query.

This structured approach ensures that VeriFed satisfies the *safety* (via EAM/DLPF) and *liveness* (via Incremental Execution) requirements defined in Section 3.

4. Snapshot-Aligned Continuous Join

The core challenge in continuous cross-chain analytics is the *temporal mismatch* problem: blockchains operate asynchronously with different block production rates and timestamping mechanisms. A naive join over the "latest" blocks often yields semantically inconsistent results (e.g., joining a payment on Chain A with a receipt on Chain B that has not happened yet).

This section details the Unified Adapter Layer (UAL) and the Snapshot Alignment Scheduler, which collectively ensure that every query epoch is derived from a rigorously synchronized global state.

4.1. Unified Adapter Layer (UAL)

The UAL acts as a translation layer that normalizes the heterogeneity of underlying blockchains into a standard relational interface. It consists of two key components: the *Finality Gadget* and the *Schema Normalizer*.

4.1.1. Finality Gadget and Abstract Block

To abstract away consensus differences, the UAL treats each blockchain C_i as a stream of Abstract Blocks. An abstract block $AB_{i,h}$ is a tuple:

$$AB_{i,h} = \langle h, t_{i,h}, \rho_{i,h}, \phi_{i,h} \rangle \tag{2}$$

where

- h is the canonical height.
- $t_{i,h}$ is the consensus timestamp (Unix epoch).
- $\rho_{i,h}$ is the *State Root* (e.g., Merkle Root) committing to the data.
- $\phi_{i,h} \in [0, 1]$ is the *Finality Confidence*.

The UAL implements chain-specific adapters to map native properties to this tuple. For BFT chains (e.g., Cosmos), $\phi_{i,h} = 1$ immediately upon commit. For probabilistic chains (e.g., Bitcoin), $\phi_{i,h} = 1 - e^{-\lambda \cdot d}$, where d is the confirmation depth. The system only exposes blocks where $\phi_{i,h} \geq 1 - \epsilon$.

4.1.2. Schema Normalizer

Different chains store data differently (e.g., EVM storage slots vs. UTXO). The Schema Normalizer maps these physical storage layouts to a virtual relational schema $R_i(A_1, \ldots, A_m)$. For example, an ERC-20 token contract is mapped to a table Transfer (from, to, value, block_ts). This allows the execution engine to issue standard SQL queries without handling low-level serialization.

4.2. Snapshot Alignment Scheduler

The scheduler is responsible for continuously producing Δ-aligned snapshot vectors $H^{(e)}$. We employ a Sliding Window Alignment algorithm to maximize the epoch throughput while satisfying the skew constraint Δ.

4.2.1. Algorithm Overview

The scheduler maintains a sliding window of finalized block headers for each chain. Let W_i be the queue of recent finalized headers for chain C_i. The algorithm proceeds as follows: Algorithm 1 summarizes the full procedure.

Algorithm 1: Sliding Window Snapshot Alignment

Input: Streams of finalized blocks from n chains; Skew bound Δ
Output: Stream of aligned vectors $H^{(e)}$

$W_1, \ldots, W_n \leftarrow \emptyset$;
$H_{last} \leftarrow \langle 0, \ldots, 0 \rangle$;
while *System is running* **do**
  foreach *chain i* **do**
    Fetch new finalized blocks and append to W_i;
  $t_{min_latest} \leftarrow \min_i(W_i.last().timestamp)$;
  `/* Try to align relative to the slowest chain's head */`
  $H_{candidate} \leftarrow \langle\rangle$;
  $valid \leftarrow$ True;
  foreach *chain j* **do**
    $b_{best} \leftarrow$ Null;
    foreach $b \in W_j$ **do**
      if $|b.timestamp - t_{min_latest}| \leq \Delta$ **then**
        $b_{best} \leftarrow b$;
    if $b_{best} \neq Null$ ***and*** $b_{best}.h > H_{last}[j]$ **then**
      $H_{candidate}.append(b_{best}.h)$;
    else
      $valid \leftarrow$ False;
      Break;
  if *valid* **then**
    $H_{last} \leftarrow H_{candidate}$;
    Emit Epoch e with $H_{candidate}$;
  `/* Prune old blocks to save memory */`
  foreach *chain i* **do**
    Remove blocks from W_i where $timestamp < t_{min_latest} - \Delta$;

1. **Candidate Selection:** Identify the chain C_{slow} with the *oldest* latest timestamp: $t_{slow} = \min_i(\max_{b \in W_i} t_b)$. This chain effectively throttles the global progress.

2. **Window Pruning:** Discard any block $b \in W_j$ where $t_b < t_{slow} - \Delta$, as it can no longer be part of a valid vector centered around t_{slow}.
3. **Vector Formation:** Attempt to construct a vector $H = \langle h_1, \ldots, h_n \rangle$ by selecting $b_j \in W_j$ such that $|t_{b_j} - t_{slow}| \leq \Delta$ for all j. To ensure determinism, if multiple blocks qualify, select the one with the largest timestamp among the finalized candidates in W_j.
4. **Epoch Emission:** If a valid H is found and $H > H^{(e-1)}$, emit epoch e. Otherwise, wait for new blocks.

4.2.2. Handling Clock Drift and Latency

VeriFed does not assume a universal clock. Since block timestamps are set by miners and may drift, the UAL applies a Median Time Adjustment. It computes a moving median of the past 11 blocks' timestamps to filter out malicious outliers. Furthermore, the system enforces a *Safe Wait Period* equal to the maximum expected network latency plus clock skew to ensure that "future" blocks do not retroactively invalidate an alignment. This guarantees that once an epoch e is emitted, it is stable and never rolled back.

4.2.3. No-Alignment Liveness Handling

If no valid Δ-aligned vector can be formed within a bounded waiting interval T_{wait}, VeriFed does not emit an unverifiable result. Instead, it emits a signed *heartbeat epoch* containing the latest verified digest pointer and a `NO_RESULT` marker, then retries alignment in the next scheduling round. This mechanism preserves safety (no misaligned query output) while providing operational liveness (the stream remains observable and auditable rather than silently stalling). Users can configure (Δ, T_{wait}) as an SLO pair: smaller Δ improves temporal strictness, while larger T_{wait} reduces heartbeat frequency under high skew.

4.2.4. Worked Multi-Epoch Example

Consider three chains (C_1, C_2, C_3) with $\Delta = 12$ s. Suppose the emitted aligned vectors are:

$$H^{(1)} = (101, 77, 55), \quad H^{(2)} = (102, 77, 56), \quad H^{(3)} = (103, 78, 56) \tag{3}$$

Only a subset changes between consecutive epochs:

$$\mathcal{C}^{(2)}_{change} = \{1, 3\}, \qquad \mathcal{C}^{(3)}_{change} = \{1, 2\} \tag{4}$$

Thus, epoch progress is monotone per chain and sparse across chains. Unchanged chains reuse their previous local roots, while changed chains trigger delta proof updates only on affected GMT paths. This directly matches the DLPF incremental update model in Section 5.1.2 and resolves the apparent mismatch between epoch semantics and sparse updates.

5. Incremental Verification

This section details the verification layer of VeriFed (see Figure 3), which ensures that the query results derived from the aligned snapshot vectors (Section 4) are authentic and tamper-evident. The core innovation is the Delta-Linked Proof Forest (DLPF) (shown in Figure 4), an authenticated data structure that supports efficient incremental updates, and the Epoch Attestation Mesh (EAM), a decentralized consensus protocol for validating these updates.

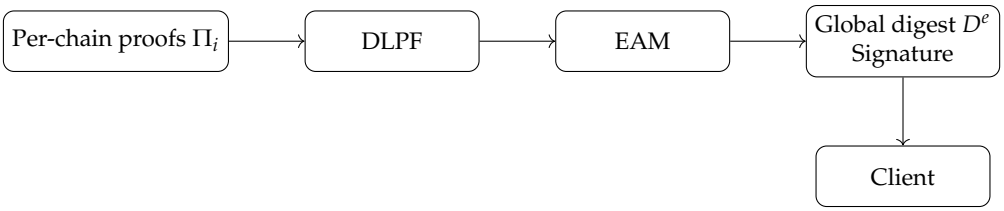


Figure 3. Verification architecture with EAM and DLPF.

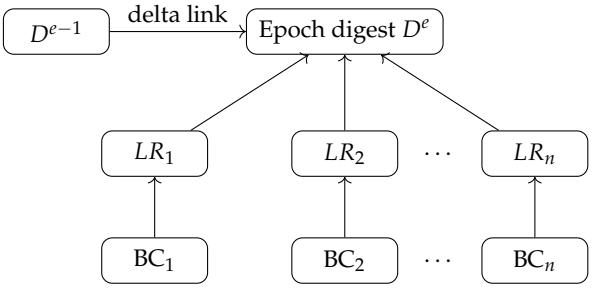


Figure 4. Delta-Linked Proof Forest (DLPF) structure.

5.1. Delta-Linked Proof Forest (DLPF)

The DLPF is a hierarchical structure designed to aggregate proofs from multiple heterogeneous chains into a single global digest D^e. Unlike traditional approaches that recompute the entire global state for every query, the DLPF exploits the fact that between consecutive epochs, only a subset of chains produce new blocks. From an information-theoretic standpoint, the DLPF minimizes the *informational redundancy* of the verification process. Since the blockchain state evolution has low *entropy rate* (most data remains static between blocks), transmitting the full state proof is inefficient. The DLPF acts as an optimal source code, transmitting only the *innovation* (state deltas) required to update the verifier's knowledge.

5.1.1. Structure Definition

The DLPF consists of two tiers:

1. **Local Tier (Per-Chain):** Each blockchain C_i maintains a local Merkle Tree MT_i over its transactions. The root $LR_i^{(e)}$ represents the finalized state of chain C_i at the aligned height $h_i^{(e)}$.
2. **Global Tier (Cross-Chain):** The system maintains a Global Merkle Tree (GMT) where the i-th leaf stores the tuple $\langle LR_i^{(e)}, h_i^{(e)}, t_i^{(e)} \rangle$. The root of the GMT is the Global Digest D^e.

The global digest D^e is a cryptographic commitment to the entire aligned global state of epoch e.

5.1.2. Incremental Update Strategy

Let $\mathcal{C}_{change}^{(e)}$ be the set of chains that have produced new finalized blocks in epoch e (i.e., $h_i^{(e)} > h_i^{(e-1)}$). For all other chains $j \notin \mathcal{C}_{change}^{(e)}$, the state remains identical ($LR_j^{(e)} = LR_j^{(e-1)}$). Instead of recomputing D^e from scratch (an $O(n)$ operation), VeriFed performs an *incremental path update*:

1. **Delta Retrieval:** The execution engine fetches the new local roots $LR_i^{(e)}$ only for $i \in \mathcal{C}_{change}^{(e)}$.
2. **Path Patching:** The EAM updates the leaf nodes for these changed chains and rehashes only the affected branches of the GMT to obtain D^e.

This reduces the computational complexity of digest generation from $O(n)$ to $O(|\mathcal{C}_{change}^{(e)}| \cdot \log n)$. To bound storage growth under long-running streams, VeriFed uses periodic checkpoints and delta pruning: only changed branches and epoch digests are retained for a sliding horizon, while older intermediate paths are compacted into checkpoint roots. Under checkpoint interval w, online storage is $O(n + w \cdot |\mathcal{C}_{change}| \log n)$ rather than linear in the total chain history. As a concrete scale example, with $n = 8$ chains, $w = 120$ epochs, and average $|\mathcal{C}_{change}| = 2$, the retained hash-state magnitude is approximately $8 + 120 \cdot 2 \cdot \log_2 8 = 728$ hash units per horizon (before proof-payload compression), which is orders smaller than retaining full per-epoch multi-chain trees.

5.2. Epoch Attestation Mesh (EAM)

The EAM is a decentralized committee of validator nodes that act as witnesses for the validity of D^e. Its primary role is to ensure that the published digest corresponds to a valid aligned snapshot vector. EAM does not replace native consensus of heterogeneous chains; instead, it verifies finalized chain-native commitments and reaches cross-chain attestation consensus over their aligned combination.

5.2.1. Verification Object and Trust Root

The exact verification object for epoch e is:

$$VO^{(e)} = \langle qid, e, H^{(e)}, D^{(e)}, \Sigma^{(e)}, hash(D^{(e-1)}) \rangle \tag{5}$$

where $\Sigma^{(e)}$ is the EAM threshold signature over $VO^{(e)}$. The trust root is two-layered: chain-native finalized roots $LR_i^{(e)}$ anchor data authenticity at source chains, and $\Sigma^{(e)}$ anchors cross-chain attestation for their aligned aggregation. Client verification accepts a result only if both anchors validate.

5.2.2. Attestation Protocol

For each epoch e, the EAM executes the following protocol: Algorithm 2 presents the incremental digest update and attestation steps.

Algorithm 2: Incremental Global Digest Update

Input: Previous Global Tree GMT^{e-1}; Aligned Vector $H^{(e)}$; Set of new Local Roots $\{LR_i\}_{i \in \mathcal{C}_{change}}$
Output: New Global Digest D^e; Signature Σ^e

$GMT^e \leftarrow GMT^{e-1}$;
foreach $i \in \mathcal{C}_{change}$ **do**
  $Leaf_i \leftarrow \langle LR_i, H^{(e)}[i].h, H^{(e)}[i].t \rangle$;
  Update leaf i in GMT^e with $Leaf_i$;
  Recompute hashes along the path from $Leaf_i$ to Root;
$D^e \leftarrow GMT^e.Root$;
`/* Consensus Phase */`
$Votes \leftarrow 0$;
foreach *Validator* $v \in EAM$ **do**
  if *VerifyAlignment*$(H^{(e)})$ ***and*** *VerifyFinality*$(\{LR_i\})$ **then**
    $Votes \leftarrow Votes + 1$;
if $Votes \geq \frac{2}{3}|EAM|$ **then**
  $\Sigma^e \leftarrow \text{ThresholdSign}(D^e, D^{e-1})$;
  return (D^e, Σ^e);
else
  return Failure;

1. **Verify Alignment:** Validators check if the snapshot vector $H^{(e)}$ satisfies the Δ-skew bound defined in Section 4.

2. **Verify Finality:** Validators query the underlying chains to confirm that each $LR_i^{(e)}$ is indeed finalized.
3. **Threshold Signature:** If the digest is valid, validators sign it using a threshold signature scheme (e.g., BLS). A valid signature Σ^e requires a 2/3 majority.
4. **Chain Linking:** The signature Σ^e includes a hash of the previous epoch's digest D^{e-1}, forming a tamper-evident chain of epochs.

5.2.3. Complexity Analysis

The incremental update strategy dramatically slashes verification overhead. While initial construction takes $O(n)$, subsequent updates only require $O(k \log n)$ operations, where $k = |C_{change}^{(e)}|$ is the number of active chains. Since typically $k \ll n$ in large-scale cross-chain environments, VeriFed scales efficiently with the number of participating blockchains.

6. Incremental Multi-Objective Optimization

Query optimization in heterogeneous cross-chain environments is uniquely challenging due to:

1. **Dual Objectives:** Users must balance *latency* (execution time) and *cost* (gas fees for on-chain verification).
2. **Dynamic Statistics:** Data distributions change continuously across epochs, rendering static plans obsolete.

To address this, we propose an Incremental Cost-Based Optimizer that adapts execution plans in real time, bypassing the prohibitive overhead of full re-optimization.

6.1. Cost Model with Gas Awareness

We define a hybrid cost model that explicitly accounts for both off-chain computation and on-chain verification expenses.

Definition 4 (Local Cost Model, LCM)**.** *For a sub-query q on blockchain BC_i, the cost is:*

$$LCM_i(q) = (1-\lambda) \cdot \underbrace{T_{exec}(q)}_{Time} + \lambda \cdot \rho \cdot \underbrace{G_{exec}(q)}_{Gas} \tag{6}$$

where

- *$T_{exec}(q)$ is the estimated I/O and CPU time for executing q off-chain.*
- *$G_{exec}(q)$ is the estimated gas required to verify the result of q on-chain (e.g., verifying Merkle proofs).*
- *ρ is the exchange rate converting gas to normalized monetary units.*
- *$\lambda \in [0, 1]$ is a user-defined preference factor (0 = minimize latency, 1 = minimize fees).*

Definition 5 (Global Cost Model, GCM)**.** *The total cost of a query plan π is the sum of local costs plus data transfer overheads:*

$$GCM(\pi) = \sum_{q \in \pi} LCM(q) + \sum_{(i,j) \in Edges(\pi)} C_{transfer}(i, j) \tag{7}$$

where $C_{transfer}(i, j)$ models the latency and fees of moving data from chain i to chain j (or to the EAM).

6.2. Incremental Plan Adaptation

Traditional optimizers (like System R) re-run the dynamic programming (DP) algorithm from scratch whenever statistics change. In our continuous setting, this is pro-

hibitively expensive ($O(3^n)$). Instead, we observe that between epochs e and $e+1$, only a subset of chains $\mathcal{C}_{change}$ update their data. This implies that for any sub-plan S that does not involve chains in $\mathcal{C}_{change}$, the optimal cost $Cost(S)$ remains valid.

6.2.1. Algorithm: Delta-DP

We propose Delta-DP, an incremental dynamic programming algorithm. It maintains the DP table $DP[S]$ (optimal cost for joining subset S) across epochs and only updates entries affected by changed statistics. Algorithm 3 is the final camera-ready pseudocode (not a placeholder).

Algorithm 3: Incremental Join-Order Optimization (Delta-DP)

Input: Blockchain set $\mathcal{B}$; Changed chains $\mathcal{C}_{change}$; Previous DP table DP_{old}
Output: Updated optimal plan π_{new}

$DP_{new} \leftarrow DP_{old}$;
`/* 1. Invalidate affected base entries */`
foreach $i \in \mathcal{C}_{change}$ **do**
  Update statistics for chain i;
  $DP_{new}[\{i\}] \leftarrow \text{RecomputeCost}(\{i\})$;
`/* 2. Propagate updates bottom-up */`
for $size = 2$ **to** $|\mathcal{B}|$ **do**
  foreach *subset* $S \subseteq \mathcal{B}$ *s.t.* $|S| = size$ **do**
    if $S \cap \mathcal{C}_{change} \neq \emptyset$ **then**
      $DP_{new}[S] \leftarrow \infty$;
      foreach *split* $L \cup R = S$ **do**
        $cost \leftarrow DP_{new}[L] + DP_{new}[R] + C_{join}(L, R)$;
        if $cost < DP_{new}[S]$ **then**
          $DP_{new}[S] \leftarrow cost$;
          $Plan[S] \leftarrow (L, R)$;
return $\text{ReconstructPlan}(Plan[\mathcal{B}])$;

6.2.2. Complexity Gain

Standard DP has complexity $O(3^n)$. Delta-DP reduces this to roughly $O(3^k \cdot 2^{n-k})$, where $k = |\mathcal{C}_{change}|$. Since typically $k \ll n$, the optimization latency is drastically reduced, enabling the system to adapt plans within the tight time budget of an epoch. The architecture of this Incremental Optimizer is depicted in Figure 5.

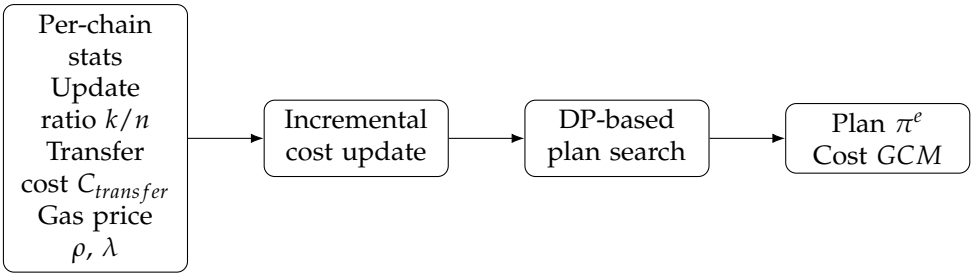


Figure 5. Optimizer architecture with incremental cost updates and DP-based planning.

7. Security Model and Proofs

In this section, we formally analyze the security properties of VeriFed. We focus on two critical guarantees: *soundness* (Tamper-Evidence) and *temporal consistency* (Skew-

Boundedness). We assume the cryptographic primitives (e.g., hash functions, digital signatures) are secure.

7.1. Security Model

We consider an adversary $\mathcal{A}$ who controls a subset of computational nodes and network links. The adversary can carry out the following:

- **Corrupt Compute Nodes:** Malicious nodes may return incorrect join results, omit tuples, or forge proofs.
- **Delay Messages:** Network adversary can delay snapshot headers or result streams up to a maximum bound Δ_{net}.
- **Collude:** Malicious nodes may collude to generate invalid aggregate signatures, bounded by the assumption that less than 1/3 of EAM validators are corrupt (BFT assumption).

We do *not* consider adversaries who can break standard cryptographic primitives (e.g., finding hash collisions).

7.2. Formal Analysis

Theorem 1 (Soundness of DLPF)**.** *Let $\mathcal{H}$ be a collision-resistant hash function. For any epoch e, if an adversary $\mathcal{A}$ generates a forged result $R' \neq R^{(e)}$ or a forged proof Π' such that Verify$(R', \Pi', D^{(e)}) =$ True, then $\mathcal{A}$ has found a collision in $\mathcal{H}$.*

Proof. The Global Digest $D^{(e)}$ is the root of the Global Merkle Tree (GMT), constructed from local roots $\{LR_i\}$. Each LR_i is the root of a Local MPT of chain C_i. If Verify$(R', \Pi', D^{(e)}) =$ True, there exists a valid Merkle path from some leaf in a local tree to $D^{(e)}$. Since $D^{(e)}$ is signed by the EAM (which requires $> 2/3$ honest votes), we assume $D^{(e)}$ correctly commits to the set of valid local roots $\{LR_i\}$. For R' to be accepted, it must match the leaf hash in the proof. If $R' \neq R^{(e)}$, then one of the following is true: 1. The leaf hash matches the authentic leaf, implying $H(R') = H(R^{(e)})$, a collision in $\mathcal{H}$. 2. The path in Π' leads to $D^{(e)}$ but differs from the authentic path, implying a collision at some intermediate node in the Merkle tree. Both cases contradict the collision resistance of $\mathcal{H}$. Thus, soundness holds. □

Theorem 2 (Temporal Consistency)**.** *Let Δ be the user-defined skew bound. If the Alignment Scheduler outputs a vector $H^{(e)} = [h_1, \ldots, h_n]$, then for any two chains i, j, the timestamp difference is $|T(h_i) - T(h_j)| \leq \Delta$.*

Proof. The Alignment Scheduler constructs $H^{(e)}$ by selecting block headers h_k from each chain k. The algorithm imposes a constraint, $\forall k, T(h_k) \in [T_{ref} - \Delta/2, T_{ref} + \Delta/2]$, where T_{ref} is the reference timestamp. Equivalently, the implementation checks the pairwise-skew form $\max_i T(h_i) - \min_i T(h_i) \leq \Delta$, which is exactly the definition in Section 2.2. For any two chains i, j, we have $|T(h_i) - T(h_j)| \leq \max_i T(h_i) - \min_i T(h_i) \leq \Delta$. Thus, the snapshot vector is Δ-aligned. The EAM validates this bound before signing $D^{(e)}$, ensuring that any vector violating the bound is rejected. □

Theorem 3 (Completeness of Incremental Verification)**.** *The incremental verification protocol accepts all valid results derived from the state transition stream $\delta^{(e)}$.*

Proof. The incremental result $R^{(e)}$ is derived from $R^{(e-1)}$ and $\delta^{(e)}$. The DLPF updates proofs only for changed paths. For any tuple $t \in R^{(e)}$, either t was unchanged (proof remains valid) or t was newly generated/modified (proof updated). Since the EAM updates $D^{(e)}$ using the same transition logic, the client's updated proof $\Pi^{(e)}$ will strictly match the new root $D^{(e)}$. Therefore, honest verification always succeeds. □

8. Experimental Evaluation

In this section, we comprehensively evaluate the performance, scalability, and security of VeriFed. Our evaluation aims to answer the following research questions:

- **RQ1 (Scalability):** How does VeriFed scale with the number of blockchains, dataset size, and consensus nodes?
- **RQ2 (Efficiency):** What is the performance gain of the incremental verification (DLPF) and optimization (Delta-DP) mechanisms compared to full recomputation and static planning?
- **RQ3 (Security):** Can VeriFed effectively detect tampering and maintain temporal consistency under adversarial conditions?
- **RQ4 (Practicality):** Is the system practical for real-world deployment on existing blockchain infrastructures?

8.1. Experimental Setup

Implementation and Environment. We implement VeriFed in Rust and evaluate it using a custom simulator on a machine with an Apple M4 CPU (10 cores), 16GB memory, and macOS 26.3. The simulator models a cross-chain environment with partitionable datasets and adjustable network delays. We use synthetic datasets (DS1–DS3) and a real-world Ethereum transaction dataset. The key parameters are summarized in Table 1. Unless otherwise stated, the results are averaged over 5 runs with error bars representing standard deviation.

Table 1. Experimental parameters and coverage of the configuration sweep.

Parameters	Variation Range
Datasets	10K (DS1), 50K (DS2), 100K (DS3)
Number of Blockchains (n)	2, 4, 6, 8
Consensus Nodes ($\|M\|$)	10, 20, 30, 40, 50
Skew Bound Δ (blocks)	3, 12, 24, 48
Epoch Length (blocks)	5, 10, 15
Update Ratio k/n	0.5%, 10%, 30%, 85%
Failure Rate	0%, 10%, 20%, 30%, 40%
Tamper Rate	0%, 10%, 20%, 40%

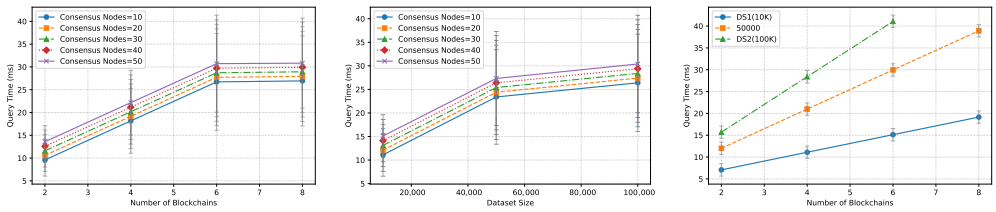
Failure and Skew Modeling. We emulate consensus overhead using Ed25519 signatures. Node failures are injected at rates from 0% to 40%. Snapshot skew is simulated by lagging non-updated chains. Tampering is simulated by flipping bytes in global digests or signatures.

Real-Data Protocol. We partition the Ethereum dataset by time-slicing and address hashing. The real-data evaluation validates that our synthetic workload accurately reflects real-world performance characteristics, with observed metrics (e.g., TPS, latency) matching closely between synthetic and real traces (see Table 4). Heterogeneous Network Setting. To avoid homogeneous-chain bias, each chain is assigned an independent delay process and block-advance process in the simulator. Reported latency curves therefore reflect asynchronous, non-identical chain behaviors rather than a uniform per-chain configuration. Code and Artifact Availability. We provide scripts for simulation, plotting, and dataset preprocessing in an anonymized artifact package for review; the full repository will be made public upon acceptance.

8.2. RQ1: Scalability Analysis

We evaluate the scalability of VeriFed by varying the number of blockchains, dataset size, and consensus nodes.

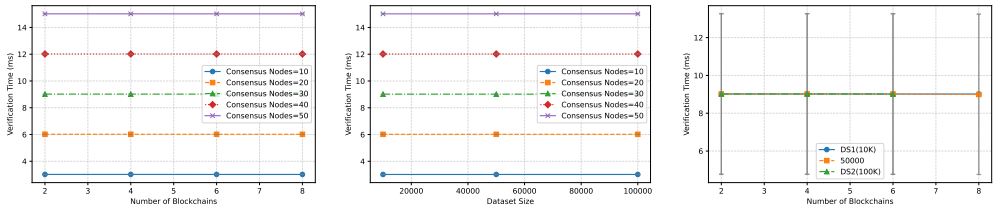
Query Performance. Figure 6 illustrates the per-epoch query latency. (a) *Impact of Chain Count:* As the number of blockchains increases from 2 to 8, latency grows linearly. This trend is primarily driven by the *straggler effect* in the UAL synchronization phase. The UAL must wait for the header of the slowest chain to effectively align the snapshot vector. This waiting time is theoretically bounded by the requirement of *temporal consistency* (Theorem 2), which mandates that all headers h_i in a snapshot must satisfy $|T(h_i) - T(h_j)| \leq \Delta$. Consequently, the alignment barrier introduces a dependency on the maximum network latency ($\max(L_1, \ldots, L_n)$) among participating chains, as the system cannot proceed until the "slowest" chain provides a valid header within the Δ window. (b) *Impact of Dataset Size:* Increasing dataset size from 10K (DS1) to 100K (DS3) has a more pronounced impact on latency. This is attributed to the computational complexity of the local join processing, which typically follows $O(N \log N)$ for Sort-Merge joins. Since VeriFed executes these joins off-chain, the dominant cost becomes the local CPU execution and memory bandwidth rather than consensus throughput. This validates our architecture of offloading compute-intensive joins to untrusted clients, verified later by lightweight proofs. (c) *Impact of Consensus Nodes:* The number of consensus nodes has a minimal impact on query latency. This confirms our design choice of decoupling execution (off-chain) from attestation (on-chain). The EAM only signs the global digest, an operation with $O(1)$ complexity relative to the dataset size, ensuring that adding validators improves security without degrading query performance. Overall, VeriFed maintains sub-second query latency for typical configurations and achieves over 60,000 processed result-tuples per second in our workload configuration (Table 4), demonstrating suitability for high-frequency continuous monitoring.



(**a**) (**b**) (**c**)

Figure 6. Query performance of VeriFed under continuous execution. (**a**) Varied blockchains and consensus nodes. (**b**) Varied dataset size and consensus nodes. (**c**) Varied blockchains and datasets.

Verification Performance. Figure 7 reports the verification cost. (a) Verification time scales logarithmically with dataset size and linearly with the number of *updated* chains. This validates the efficiency of the DLPF, which allows clients to verify only the Δ (changes) rather than the full state history. Specifically, the verifier only needs to check the Merkle branches corresponding to the modified tuples, avoiding the $O(N)$ cost of scanning the entire tree. This result empirically supports Theorem 3 (Completeness of Incremental Verification), confirming that partial proofs are sufficient for correctness. (b) The EAM size introduces a slight overhead due to signature aggregation, but this is negligible compared to proof verification. (c) Even with 8 chains and 100K records, verification completes within 100 ms. This efficiency is critical for stream processing, ensuring that the verification layer does not become a bottleneck that stalls the consumption of query results.



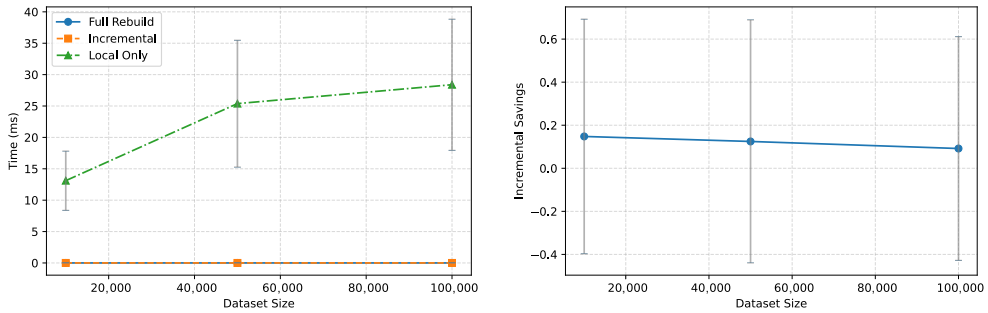
(**a**) (**b**) (**c**)

Figure 7. Verification performance of VeriFed under continuous execution. (**a**) Varied blockchains and consensus nodes. (**b**) Varied dataset size and consensus nodes. (**c**) Varied blockchains and datasets.

8.3. RQ2: Efficiency of Incremental Verification

We quantify the benefits of the Delta-Linked Proof Forest (DLPF) and Delta-DP optimizer. From the perspective of information theory, the verification process can be viewed as transmitting the state update to the client. The *entropy* of the state update $H(\delta)$ is proportional to the number of changed tuples. A full verification scheme has a cost $O(N)$, which is highly redundant when $H(\delta) \ll N$. The DLPF achieves a cost $O(H(\delta) \cdot \log N)$, effectively compressing the proof to the information-theoretic limit of the update stream.

Incremental Verification Savings. Figure 8 compares VeriFed's incremental verification against a "Full Rebuild" baseline (recomputing the global Merkle root from scratch) and a "Local Only" baseline (verifying n independent proofs). (a) VeriFed achieves up to 80% savings in verification time when the update ratio k/n is low (e.g., 10%). The "Full Rebuild" approach requires re-hashing all leaf nodes ($O(N)$), whereas DLPF only re-hashes the path from the modified leaves to the root ($O(k \log N)$). This algorithmic advantage explains the massive gap at low update ratios and is a direct consequence of the *soundness* property (Theorem 1), which ensures that un-updated branches do not need to be re-verified. (b) The savings are consistent across dataset sizes, confirming that DLPF effectively exploits the temporal locality of blockchain state changes. While the "Local Only" approach (verifying n independent signatures) has lower end-to-end latency due to skipping consensus (as shown in Table 3), its *client-side* verification complexity scales linearly ($O(n)$) and lacks the global consistency guarantees provided by VeriFed's aggregated proof. VeriFed's EAM signature aggregation ensures that client verification cost remains nearly constant regardless of the chain count.



(**a**) (**b**)

Figure 8. Impact of incremental verification (DLPF) on performance. (**a**) Verification time comparison. (**b**) Incremental savings.

Optimization Efficiency. We compared our Delta-DP optimizer (Algorithm 3) against Fixed Order and Greedy strategies. Fixed Order often resulted in 2–5× higher costs due to suboptimal join orders. Greedy approached DP performance for simple star schemas but degraded significantly for complex topologies. Delta-DP maintained near-optimal

plans while reducing optimization time by an order of magnitude (typically from seconds to milliseconds) compared to full DP re-optimization. This is because Delta-DP avoids re-exploring the entire join order search space ($O(3^n)$) and instead only updates the cost entries for the sub-plans affected by the changed chains ($O(3^k \cdot 2^{n-k})$). This makes it feasible to run sophisticated cost-based optimization within the short inter-block interval without stalling the pipeline.

Gas Preference Parameter λ. The parameter λ controls the latency–cost trade-off in Equation (6). Operationally, λ is configured per query policy profile: latency-critical profiles use lower λ, while cost-sensitive profiles use higher λ. In deployment, we use a three-level policy: latency-first ($\lambda \in [0.1, 0.3]$), balanced ($\lambda \in [0.3, 0.6]$), and cost-first ($\lambda \in [0.6, 0.9]$). To prevent unstable plan oscillation under gas volatility, we use epoch-to-epoch smoothing of gas input and only trigger plan replacement when the expected gain exceeds a minimum improvement threshold. This ensures that the optimizer reacts to meaningful market shifts rather than noise.

Proof/Bandwidth/Storage Overhead Reporting. Beyond latency and throughput, we report protocol-level overhead dimensions:

$$|\Pi^{(e)}| \approx |\Sigma^{(e)}| + \sum_{i \in \mathcal{C}_{change}^{(e)}} \left(|\pi_i^{local}| + |\pi_i^{gmt}| \right) \tag{8}$$

where $|\pi_i^{local}| = O(\log N_i)$ and $|\pi_i^{gmt}| = O(\log n)$. Therefore, proof size and verification bandwidth scale with changed chains rather than all chains. Combined with checkpoint compaction (Section 5.1.2), online storage remains bounded by the sliding horizon rather than total chain history.

Initialization Overhead. Figure 9 shows that preprocessing (building initial Merkle structures and EAM setup) scales linearly with system size but remains below 2 s for the largest configuration. This one-time cost is amortized over thousands of epochs in a continuous query.

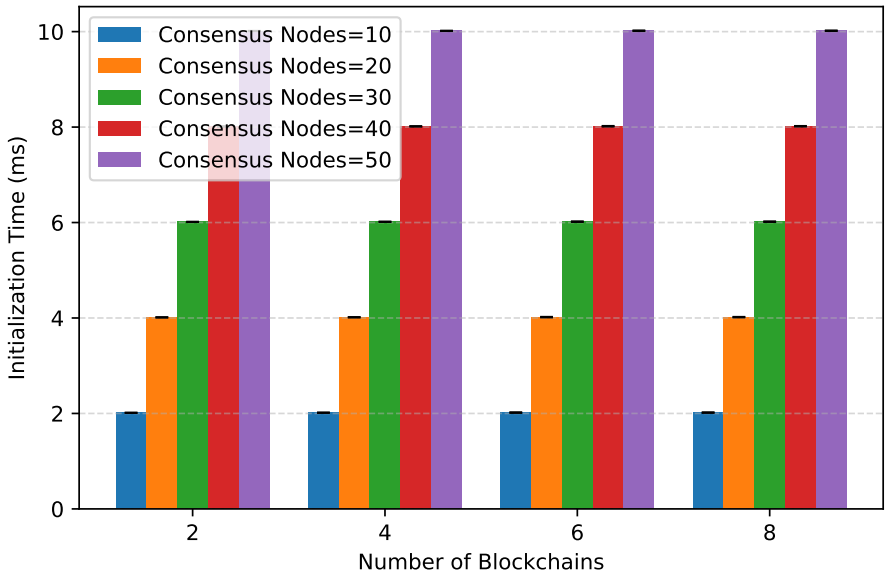
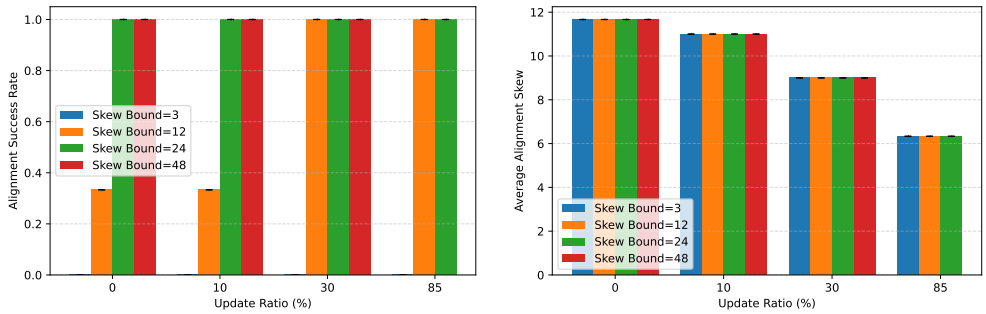


Figure 9. Preprocessing overhead for building the DLPF and forming the EAM.

8.4. RQ3: Security and Integrity

Temporal Alignment. Figure 10 shows the alignment success rate under varying skew bounds Δ. We observe a fundamental trade-off: Tighter bounds (e.g., $\Delta = 3$) increase the

risk of alignment failure when network latency is high, but ensure stricter temporal consistency. Relaxing Δ improves the success rate but allows larger temporal drift between joined tuples. The strong negative correlation (-0.581) between Δ and skew violations confirms that the Alignment Scheduler effectively enforces the user's consistency requirements. In practice, a bound of $\Delta = 12$ (approx. 2 min on Ethereum) offers a sweet spot, balancing high success rates (>95%) with acceptable temporal divergence for most analytical queries.



(**a**) (**b**)

Figure 10. Alignment success and staleness under varying update ratios. (**a**) Alignment success rate. (**b**) Average alignment skew.

Tamper Detection. Figure 11 demonstrates VeriFed's robustness against active attacks. We injected invalid proofs and modified global digests. The system detected 100% of invalid proofs and signatures, with zero false positives. This high detection rate is guaranteed by the *soundness* property (Theorem 1), which relies on the collision resistance of the Merkle structure and the unforgeability of the EAM threshold signatures. Furthermore, the detection rate for subtle replay attacks (using stale but valid proofs from past epochs) was also near 100%. This validates the effectiveness of the *epoch-linked* structure of the DLPF, which binds proofs to a specific temporal epoch ID, preventing adversaries from presenting historically valid states as current ones.

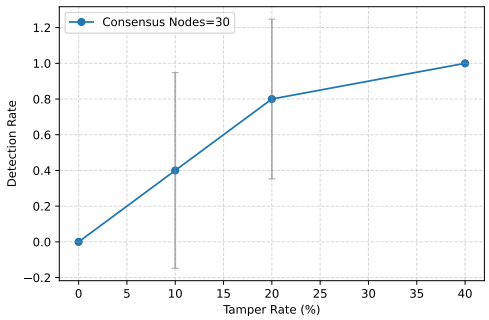


Figure 11. Tamper detection rate under varying tamper rates.

8.5. RQ4: Practicality and Resilience

System Resilience. Figure 12 shows system throughput and latency under node failures. VeriFed maintains stable throughput up to a 30% failure rate. This resilience is a direct consequence of the BFT consensus protocol used in the EAM, which tolerates up to $f < |M|/3$ faulty nodes. Beyond 33% failures, performance degrades as expected due to repeated view changes and timeouts. In the context of the CAP theorem, VeriFed favors *Consistency* and *Partition Tolerance* over Availability, ensuring that no invalid results are accepted even when the network is partially partitioned.

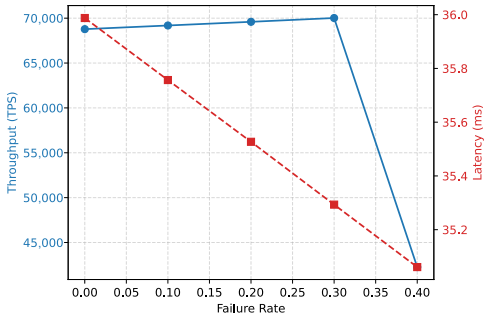
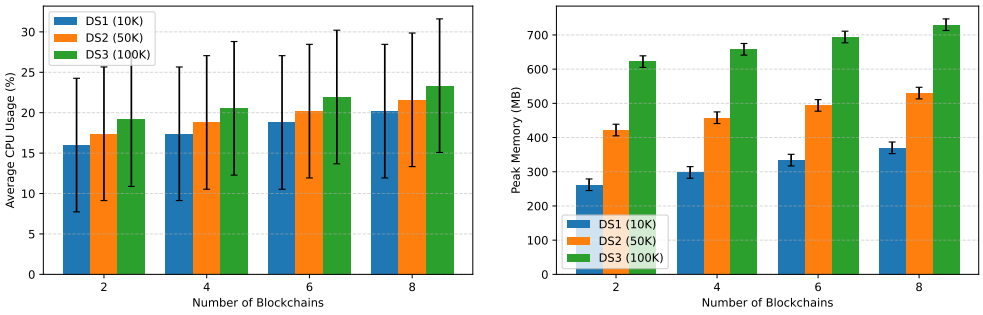


Figure 12. System throughput and latency under varying node failure rates.

Resource Consumption. Figure 13 indicates that CPU usage scales gently with the number of chains, while memory usage is dominated by dataset size. Peak memory remains within 16 GB even for the largest configuration, confirming that VeriFed is deployable on commodity hardware without requiring specialized trusted hardware (SGX). The memory footprint is largely due to the caching of Merkle tree nodes, which can be further optimized using on-disk storage if needed.



(**a**) (**b**)

Figure 13. Resource consumption across system scales. (**a**) Average CPU usage. (**b**) Peak memory.

Gas-Trace Replay Sensitivity. To directly test optimizer behavior under volatile fee markets, we add a dedicated replay experiment driven by real Ethereum gas traces from our dataset (`ethereum_transactions_sample.csv`). We construct a per-block gas series (52 replay epochs, median gas = 99,535,226,804 Wei), map it to the dynamic ρ term in Equation (6), and replay Delta-DP plan selection with smoothing and switch-threshold controls. We report two metrics: average normalized objective and plan-switch rate. Table 2 shows sensitivity over λ, ρ scaling, smoothing factor α, and plan-change threshold τ. The results show stable behavior in the default regime (zero switch oscillation at $\alpha = 0.3, \tau = 0.03$), while aggressive threshold settings ($\tau = 0$) increase switching as expected.

Baseline Comparison. We compare VeriFed with single-chain verifiable query systems (vChain, vChain++, GEM^2-Tree) adapted for cross-chain settings. The adaptation rule is explicit and uniform: each baseline executes on each chain independently, then merges outputs at an external coordinator without a global aligned epoch certificate. As shown in Figure 14 and Table 3, VeriFed delivers substantially better tail behavior than most verifiable baselines while adding explicit temporal-consistency guarantees. For example, VeriFed's P95 is 71.32 ms versus vChain's 4356.51 ms. SEBDB is slightly lower in P95 in this specific setup (71.04 ms), but it does not provide a signed cross-chain aligned snapshot object. Thus, the key advantage of VeriFed is not only latency, but *latency with verifiable cross-chain temporal consistency*.

Table 2. Gas-trace replay sensitivity (real Ethereum trace, 52 replay epochs). Objective is normalized Equation (6) cost.

Sweep	Setting	Objective/Switch
λ	0.1	1.1709/1.92%
λ	0.3	1.3998/0.00%
λ	0.5	1.4454/1.92%
λ	0.7	1.2925/1.92%
λ	0.9	1.1532/1.92%
ρ scale	0.5	1.1120/3.85%
ρ scale	1.0	1.4198/0.00%
ρ scale	1.5	1.6942/1.92%
ρ scale	2.0	1.9172/1.92%
α	0.1	1.3880/0.00%
α	0.3	1.4198/0.00%
α	0.5	1.4302/0.00%
α	0.8	1.4358/0.00%
τ	0.00	1.4196/3.85%
τ	0.01	1.4198/0.00%
τ	0.03	1.4198/0.00%
τ	0.05	1.4198/0.00%

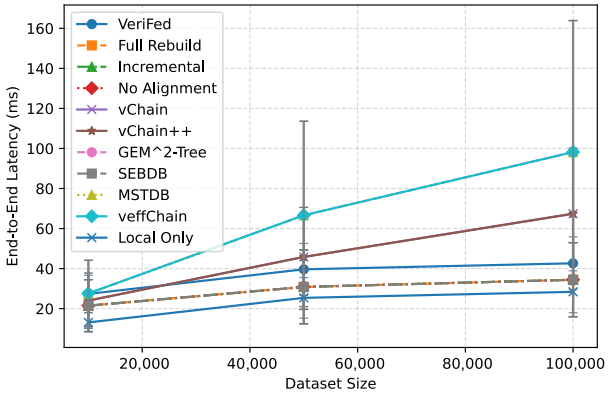


Figure 14. End-to-end latency comparison across ablations and public baselines.

Table 3. End-to-end latency percentiles (ms) for VeriFed, ablations, and top-tier baselines. Note that "Local Only" is a non-secure baseline that skips global consensus.

Method	P50	P95
VeriFed	43.21	71.32
vChain	140.13	4356.51
vChain++	110.90	2602.52
GEM^2-Tree	55.85	149.17
SEBDB	42.94	71.04
MSTDB	55.85	149.17
veffChain	55.86	149.18
Local Only	8.10	17.70

The "60,000 TPS" statement in this paper refers to the same metric as in Table 4, i.e., processed result-tuples per second under identical epoch duration, query template, and verification settings.

Table 4. Real vs. synthetic means for core metrics. TPS denotes processed result-tuples per second under the same epoch and query configuration.

Metric	Synthetic	Real
VeriFed Latency (ms)	38.34	38.32
TPS	64,036	64,077
Tamper Detection Accuracy	1.00	1.00
Alignment Success Rate	0.80	0.80

9. Related Work

9.1. Entropy in Distributed Ledgers

Recent studies have begun to apply concepts from information theory to analyze blockchain systems. Zhang et al. [16] explored the entropy-based perspective of blockchain technology, analyzing how consensus mechanisms contribute to system order. Recent work further discusses entropy and stability dynamics in blockchain systems [17]. Shannon's original mathematical theory of communication [18] provides a foundational framework for understanding information transmission limits in such distributed networks. Furthermore, Pincus [19] introduced approximate entropy as a measure of system complexity, which Assaf et al. [20] adapted to study the influence of participation on the entropy of permissionless blockchains. In VeriFed, we extend these concepts to model the temporal consistency of cross-chain data federation, treating state misalignment as a source of entropy that must be minimized.

9.2. Verifiable Query Processing on Blockchains

Recent years have seen a surge in verifiable query processing on blockchains. vChain [13] and vChain+ [15] pioneered verifiable boolean range queries using cryptographic accumulators. GEM^2-Tree [14] and MSTDB [21] optimize gas costs for range and semantic queries using hybrid storage and novel Merkle structures. Others have extended this to keyword search [22,23], graph queries [24,25], and freshness authentication [26]. However, these systems are predominantly single-chain-centric. They assume a single, consistent global state and do not address the challenge of synchronizing and querying across multiple, heterogeneous ledgers with diverging block times.

9.3. Cross-Chain Interoperability and Consistency

Cross-chain interoperability is critical for the fragmented blockchain landscape [8]. Solutions like Polkadot [10] and Cosmos [11] use relay chains and inter-blockchain communication (IBC) protocols to facilitate message passing and asset transfers. Layer-2 scaling solutions like Arbitrum [5] and Optimism [4] use rollups to offload execution while inheriting L1 security. While these protocols enable transactional interoperability, they do not support analytical interoperability. They lack mechanisms to construct a temporally consistent global snapshot of data residing on different chains for complex join processing, often requiring ad hoc, mostly centralized indexers like The Graph [12].

9.4. Distributed and Secure Join Optimization

Join processing is a classical problem in database research [27,28]. Distributed join techniques [29,30] focus on minimizing data shuffling and optimizing plan execution on parallel clusters. In the security domain, oblivious joins [31] and secure multi-party computation (SMPC) [32] protect data privacy but incur high computational overheads, making them unsuitable for low-latency continuous queries. VeriFed adopts a different trade-off: it prioritizes verifiability and performance over confidentiality (targeting public

data), using a trusted execution environment (TEE) or cryptoeconomic game (like our EAM) to ensure integrity without the heavy cost of cryptographic privacy for every operation.

9.5. Summary of Differences

VeriFed occupies a unique design space. Unlike single-chain query engines, it handles heterogeneity and temporal skew via the UAL and Δ-aligned snapshot vector. Unlike general cross-chain bridges, it provides a structured, query-able view of global state. And unlike traditional distributed databases, it embeds cryptographic verification into the execution pipeline (DLPF) to allow untrusted compute nodes to prove correctness to lightweight clients. Table 5 summarizes these distinctions.

Table 5. Capability comparison (Y = yes, N = no).

System	Cross-Chain	Temporal Alignment	Global Proof
vChain/vChain++	N	N	N
GEM^2-Tree/SEBDB/MSTDB	N	N	N
veffChain	N	N	N
VeriFed	Y	Y	Y

10. Conclusions

This paper presents VeriFed, a pioneering framework enabling verifiable, temporally consistent, and continuous join queries across heterogeneous blockchains. We formalized the problem of cross-chain temporal consistency and proposed the Δ-aligned snapshot vector model to bound state skew. To support continuous execution, we designed the Delta-Linked Proof Forest (DLPF) and Epoch Attestation Mesh (EAM), which collectively reduce client-side verification cost from linear to logarithmic complexity. Furthermore, our Delta-DP optimizer adapts join plans incrementally, ensuring low-latency processing under dynamic network conditions. Extensive evaluations using synthetic and real-world Ethereum datasets confirm that VeriFed achieves sub-second per-epoch latency (approx. 38 ms) and linear scalability, while robustly detecting tampering and enforcing strict temporal alignment. Future work will explore privacy-preserving query processing via zero-knowledge proofs and optimize data transfer for wide-area deployments.

Author Contributions: Conceptualization, K.H. and M.B.; methodology, K.H., M.B. and Y.M.; software, K.H.; validation, K.H. and M.B.; formal analysis, K.H. and Y.M.; investigation, K.H.; resources, M.B. and Y.M.; data curation, K.H.; writing—original draft preparation, K.H.; writing—review and editing, K.H., M.B. and Y.M.; visualization, K.H.; supervision, M.B. and Y.M.; project administration, M.B.; funding acquisition, M.B. and Y.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by the National Natural Science Foundation of China (No. 62402326), Fundamental Scientific Research Project of Higher Education Institutions of Liaoning Provincial Department of Education (NO. LJ212410142146), Joint Fund General Project of Liaoning Provincial Department of Science and Technology (NO. 2024-BSLH-165), and Open Project of the State Key Laboratory of Synthetical Automation for Process Industries (No. SAPI-2025-KFKT-06).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The simulation scripts, configuration files, and processed dataset split protocol are available in an anonymized review artifact and will be publicly released upon acceptance.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Dinh, T.T.A.; Liu, R.; Zhang, M.; Chen, G.; Ooi, B.C.; Wang, J. Untangling blockchain: A data processing view of blockchain systems. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 1366–1385. [CrossRef]
2. Buterin, V. A Next-Generation Smart Contract and Decentralized Application Platform. Whitepaper. 2014. Available online: https://ethereum.org/en/whitepaper/ (accessed on 10 March 2026).
3. Yakovenko, A. Solana: A New Architecture for a High Performance Blockchain v0.8.13. Whitepaper. 2018. Available online: https://solana.com/solana-whitepaper.pdf (accessed on 10 March 2026).
4. Optimism, P. Optimism: A Scalable Implementation of the Optimistic Rollup Protocol. Whitepaper. 2021. Available online: https://community.optimism.io/docs/protocol/overview/ (accessed on 10 March 2026).
5. Kalodner, H.; Goldfeder, S.; Chen, X.; Weinberg, S.M.; Felten, E.W. Arbitrum: Scalable, private smart contracts. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1353–1370.
6. Werner, S.M.; Perez, D.; Gudgeon, L.; Klages-Mundt, A.; Harz, D.; Knottenbelt, W.J. SoK: Decentralized Finance (DeFi). *arXiv* **2021**, arXiv:2101.08778.
7. Saberi, S.; Kouhizadeh, M.; Sarkis, J.; Shen, L. Blockchain technology and its relationships to sustainable supply chain management. *Int. J. Prod. Res.* **2019**, *57*, 2117–2135. [CrossRef]
8. Belchior, R.; Vasconcelos, A.; Guerreiro, S.; Correia, M. A survey on blockchain interoperability: Past, present, and future trends. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–41. [CrossRef]
9. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
10. Wood, G. Polkadot: Vision for a Heterogeneous Multi-Chain Framework. 2016. Available online: https://polkadot.network/whitepaper/ (accessed on 10 March 2026).
11. Kwon, J.; Buchman, E. Cosmos: A Network of Distributed Ledgers. Whitepaper. 2019. Available online: https://cosmos.network/ (accessed on 10 March 2026).
12. Talbot, Y.; Ramirez, B.; Pohlmann, J. The Graph: A Decentralized Query Protocol for Blockchains. Whitepaper. 2019. Available online: https://thegraph.com/docs/en/ (accessed on 10 March 2026).
13. Xu, C.; Zhang, C.; Xu, J. vchain: Enabling verifiable boolean range queries over blockchain databases. In Proceedings of the 2019 International Conference on Management of Data, Amsterdam, The Netherlands, 30 June–5 July 2019; pp. 141–158.
14. Zhang, C.; Xu, C.; Xu, J.; Tang, Y.; Choi, B. GEM^2-Tree: A Gas-Efficient Structure for Authenticated Range Queries in Blockchain. In Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macau, China, 8–11 April 2019; pp. 842–853.
15. Wang, H.; Xu, C.; Zhang, C.; Xu, J.; Peng, Z.; Pei, J. vChain+: Optimizing verifiable blockchain boolean range queries. In Proceedings of the 2022 IEEE 38th International Conference on Data Engineering (ICDE), Kuala Lumpur, Malaysia, 9–12 May 2022; pp. 1927–1940.
16. Liu, F.; Fan, H.-Y.; Qi, J.-Y. Blockchain technology, cryptocurrency: Entropy-based perspective. *Entropy* **2022**, *24*, 557. [CrossRef] [PubMed]
17. Anagnostakis, A.G.; Glavas, E. Entropy and Stability in Blockchain Consensus Dynamics. *Information* **2025**, *16*, 138. [CrossRef]
18. Shannon, C.E. A mathematical theory of communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423. [CrossRef]
19. Pincus, S.M. Approximate entropy as a measure of system complexity. *Proc. Natl. Acad. Sci. USA* **1991**, *88*, 2297–2301. [CrossRef] [PubMed]
20. Assaf, A.; Bilgin, M.H.; Demir, E. Using transfer entropy to measure information flows between cryptocurrencies. *Phys. A Stat. Mech. Its Appl.* **2022**, *586*, 126462. [CrossRef]
21. Zhou, E.; Hong, Z.; Xiao, Y.; Zhao, D.; Pei, Q.; Guo, S.; Akerkar, R. MSTDB: A hybrid storage-empowered scalable semantic blockchain database. *IEEE Trans. Knowl. Data Eng.* **2022**, *35*, 8228–8244. [CrossRef]
22. Cui, N.; Wang, D.; Li, J.; Zhu, H.; Yang, X.; Xu, J.; Cui, J.; Zhong, H. Enabling efficient, verifiable, and secure conjunctive keyword search in hybrid-storage blockchains. *IEEE Trans. Knowl. Data Eng.* **2023**, *36*, 2445–2460. [CrossRef]
23. Wu, H.; Düdder, B.; Wang, L.; Cao, Z.; Zhou, J.; Feng, X. Survey on secure keyword search over outsourced data: From cloud to blockchain-assisted architecture. *ACM Comput. Surv.* **2023**, *56*, 1–40. [CrossRef]
24. Li, S.; Zhang, Z.; Xiao, J.; Zhang, M.; Yuan, Y.; Wang, G. Authenticated Keyword Search on Large-Scale Graphs in Hybrid-Storage Blockchains. In Proceedings of the 2024 IEEE 40th International Conference on Data Engineering (ICDE), Utrecht, The Netherlands, 13–16 May 2024; pp. 1958–1971.
25. Wu, H.; Li, Z.; Song, R.; Xiao, B. Enabling privacy-preserving and efficient authenticated graph queries on blockchain-assisted clouds. *IEEE Trans. Knowl. Data Eng.* **2023**, *35*, 9535–9548. [CrossRef]
26. Liu, Q.; Peng, Y.; Tang, Z.; Jiang, H.; Wu, J.; Wang, T.; Peng, T.; Wang, G. veffchain: Enabling freshness authentication of rich queries over blockchain databases. *IEEE Trans. Knowl. Data Eng.* **2023**, *36*, 2285–2300. [CrossRef]

27. Leis, V.; Gubichev, A.; Mirchev, A.; Boncz, P.; Kemper, A.; Neumann, T. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* **2015**, *9*, 204–215. [CrossRef]
28. Lan, H.; Bao, Z.; Peng, Y. A Survey on Advancing the DBMS Query Optimizer: Cardinality Estimation, Cost Model, and Plan Enumeration. *Data Sci. Eng.* **2021**, *6*, 86–101. [CrossRef]
29. Chu, X.; Balazinska, M.; Suciu, D. From Theory to Practice: Efficient Join Query Evaluation in a Parallel Database System. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, VIC, Australia, 31 May–4 June 2015; pp. 63–78.
30. Neumann, T. Query simplification: Graceful degradation for join-order optimization. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, Providence, RI, USA, 29 June–2 July 2009; pp. 403–414.
31. Chang, Z.; Xie, D.; Wang, S.; Li, F. Towards Practical Oblivious Join. In Proceedings of the 2022 International Conference on Management of Data, Osaka, Japan, 12–17 June 2022; pp. 803–817.
32. Wang, Y.; Yi, K. Secure Yannakakis: Join-Aggregate Queries over Private Data. In Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data, Xi'an, China, 20–25 June 2021; pp. 1969–1981.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.