Tsinghua Science and Technology
Doi:10.26599/TST.2026.9010087

Research Article

AdaH-Chain: Deterministic capacity-aware GNNs sharding in heterogeneous blockchains

Ziyao Wang, Yongjiao Sun, Hengtai Zhao, Yishu Wang(✉), Hangxu Ji, Xiangguo Zhao, and Jinchuan Chen

Abstract Modern blockchain sharding protocols increasingly deploy Graph Neural Networks (GNNs) to optimize state partitioning. However, executing these learning heuristics across heterogeneous networks introduces two fundamental challenges. First, hardware-dependent floating-point arithmetic evaluates inconsistently across disparate instruction set architectures. This inconsistency triggers arithmetic state divergence and permanent consensus failures. Second, existing methods enforce uniform load distribution across shards. This ignores physical hardware disparities and forces powerful nodes to idle while waiting for resource-constrained validators, creating severe straggler bottlenecks. To address these challenges, we propose AdaH-Chain, a deterministic capacity-aware sharding framework. First, we design a Cross-ISA Deterministic Execution Engine. It replaces unsafe floating-point operations with a symmetric fixed-point quantization mechanism to mathematically guarantee bit-wise consensus safety. Second, we introduce a Capacity-Aware GNN Sharding algorithm. This mechanism maps extracted transaction subgraphs proportionally to verified hardware throughput limits, effectively eliminating the straggler effect. We evaluate AdaH-Chain using real transaction traces containing one million accounts from the Ethereum Mainnet. Experimental results confirm that the fixed-point engine preserves 94.7% of the original topological clustering accuracy while ensuring a 0% consensus divergence rate. By structurally aligning workloads with hardware capacities, AdaH-Chain delivers a peak throughput of 4,800 TPS, achieving a 1.9× gain over state-of-the-art learning protocols.

Keywords Blockchain Sharding, Heterogeneous, GNNs, Determinism, Capacity-Aware Scheduling

• Ziyao Wang, Yongjiao Sun, Hengtai Zhao, Yishu Wang and Hangxu Ji are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110169,China. E-mail: wangziyao@stumail.neu.edu.cn; sunyongjiao@mail.neu.edu.cn; zhaohengtai@stumail.neu.edu.cn; wangyishu@mail.neu.edu.cn; jihangxu@mail.neu.edu.cn;
• Xiangguo Zhao is with the Software College, Northeastern University, Shenyang 110169,China. E-mail:zhaoxiangguo@mail.neu.edu.cn
• Jinchuan Chen is with the School of Information, Renmin University of China, Beijing, 100872, China. E-mail: jcchen@ruc.edu.cn
* To whom correspondence should be addressed.
* wangziyao@stumail.neu.edu.cn
Manuscript received: 20xx-xx-xx; revised: 20xx-11-xx; accepted: 20xx-

1 Introduction

To optimize state partitioning, modern blockchain sharding protocols have evolved from simple randomized assignments to intelligent learning heuristics. One mainstream approach leverages Deep Reinforcement Learning (DRL) [1, 2] to dynamically allocate accounts across shards. However, because these models evaluate macro-level workload queues rather than deep structural dependencies, they often fail to preserve graph locality, leading to high cross-shard communication overhead. Another emerging paradigm utilizes Graph Neural Networks (GNNs) [3, 4] to capture high-dimensional account dependencies and partition densely connected communities. Although this GNN strategy successfully minimizes cross-shard transactions, deploying it across heterogeneous validator networks introduces two fundamental challenges: execution consistency and secure capacity profiling.

(1) The first challenge is ensuring absolute execution consistency across heterogeneous runtimes. Mainstream smart contract execution environments (e.g., the Ethereum Virtual Machine) strictly prohibit native floating-point operations to guarantee absolute determinism across the network [5, 6]. Developers must typically execute financial logic using high-bit integer scaling to avoid execution ambiguity. However, the emerging GNN-based sharding protocols attempt to deploy complex learning heuristics into the consensus layer for topology optimization. These models inherently depend on high-dimensional tensor aggregations and non-linear activation functions that are processed by hardware-level floating-point units. Standard floating-point operations evaluate inconsistently across disparate Instruction Set Architectures (e.g., x86 versus ARM) due to varying rounding rules and fused multiply-add (FMA) execution orders [7]. In a decentralized consensus environment, even a single-bit arithmetic deviation in the sharding map calculation would lead to state root mismatches and catastrophic network splits [8]. Existing protocols lack a deterministic mechanism to reconcile the continuous nature of GNNs with the discrete, bit-perfect requirements of blockchain consensus. The left part of Fig.1 illustrates how this hardware-induced divergence causes consensus collapse.

(2) The second challenge is accurately profiling hardware constraints to prevent performance bottlenecks. Most protocols [9, 10] incorrectly assume that all hardware is identical and distribute accounts uniformly across shards. In mixed clusters ranging from resource-constrained ARM devices to high-end servers, this uniform workload inevitably triggers the straggler effect [11, 12]. During state replication, powerful nodes complete tasks rapidly, only to idle while waiting for slower validators. Solving this requires building physical hardware limits directly into the routing logic. However, current mechanisms [1, 9] lack capacity-aware capabilities. Furthermore, in a trustless network, relying on self-reported hardware metrics is insecure, as nodes may report fake capacities to serve their own interests. Traditional CPU benchmarks [13] also fail to reflect the severe storage I/O bottlenecks during state execution [14]. Thus, objectively measuring and verifying multi-dimensional capacities (CPU and I/O) becomes essential for optimal load distribution. The right part of Fig.1 illustrates this profiling constraint.

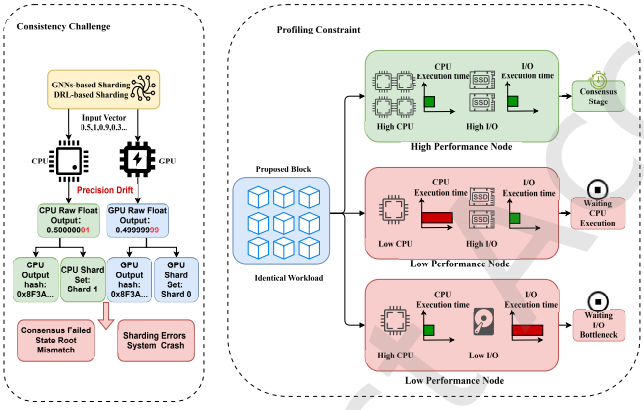


Fig. 1 Fundamental challenges in heterogeneous sharding.

To resolve these conflicts, we propose **AdaH-Chain** (Adaptive Deterministic and Heterogeneous-aware Sharding), a capacity-aware deterministic GNN sharding framework. It enforces strict blockchain determinism across diverse physical hardware. The core contributions of this work directly address the aforementioned vulnerabilities through a unified architecture.

(1) We build a Cross-ISA Deterministic Execution Engine to directly address the consistency challenge. We replace hardware-dependent floating-point operations with a symmetric fixed-point quantization mechanism. This engine entirely eliminates arithmetic divergence across mixed CPU-GPU clusters. Evaluations show this Byzantine consistency incurs minimal topological accuracy loss, preserving over 94.7% of the continuous clustering precision and adding less than a 1.5% penalty to the final edge-cut ratio.

(2) We design a Capacity-Aware GNN Sharding algorithm, coupled with a secure capacity profiling mechanism, to resolve the hardware bottleneck. AdaH-Chain discards the equal shard size assumption and optimizes for capacity utilization. The protocol strictly maps dense transaction subgraphs to verified hardware throughput ceilings. By saturating the computational limits of powerful nodes, the system effectively eliminates the straggler effect and achieves throughput gains.

(3) We comprehensively evaluate the framework using historical Ethereum transaction traces. Experimental results indicate that AdaH-Chain maintains a Cross-Shard Transaction (CST) ratio of approximately 45%. The framework delivers a 1.9× throughput improvement over the randomized sharding baseline. Experimental results validate that mapping logical shards to physical hardware limits enables highly effective performance scaling in heterogeneous environments while strictly preserving consensus safety.

2 Preliminaries and Problem Formulation

In this section, we first introduce the background of learning-driven blockchain sharding and its inherent determinism issues. Then, we formally define the system model, threat model, and the overarching problem statement. The frequently used notations are summarized in Table 1.

2.1 Graph Representation and GNN-driven Sharding

To increase scalability, state sharding partitions the global blockchain state into disjoint subgraphs, confining raw transaction processing to local shards. In this paper, we model the blockchain state as a directed and weighted transaction graph $\mathcal{G} = (A, E)$, where A represents the set of accounts and E denotes the set of transactional interactions. For any account $u \in A$, its topological neighborhood is defined as $\mathcal{N}(u) = \{v \in A | (u, v) \in E\}$, with its interaction degree denoted by $deg(u) = |\mathcal{N}(u)|$.

Recent advancements leverage Graph Neural Networks (GNNs) to compute the state routing map. A standard GNN layer updates the feature vector $h_u^{(l)}$ of account u by aggregating its neighbors' features:

$$h_u^{(l+1)} = \sigma\Big(W^{(l)} \cdot \mathrm{AGG}\big(\{h_v^{(l)} : v \in \mathcal{N}(u)\}\big)\Big) \quad (1)$$

where $W^{(l)}$ is the trainable weight matrix, $\mathrm{AGG}(\cdot)$ is the aggregation function, and $\sigma(\cdot)$ is the non-linear activation function.

The Determinism Paradox: Evaluating Eq. (1) heavily relies on hardware-accelerated floating-point arithmetic (FPA). However, in a State Machine Replication (SMR) environment, microscopic floating-point deviations, stemming

from different instruction set architectures (ISAs), evaluate non-deterministically. Such hardware-induced arithmetic ambiguity contradicts the strict determinism required by blockchain consensus.

2.2 System Model and Threat Model

System Model: Our system models a heterogeneous permissioned blockchain network operating in discrete, consecutive time intervals called *epochs*. The network consists of two primary roles: validators and leaders. Validators are responsible for parallel subgraph processing and maintaining local ledgers. The leaders are structurally divided into two tiers: the *global epoch leader*, which computes the system-wide capacity-aware sharding map at epoch boundaries, and *local shard leaders* (e.g., PBFT primaries), which aggregate intra-shard state updates.

Unlike traditional uniform models, we explicitly consider hardware heterogeneity. Each validator node i possesses a distinct capacity profile $\mathcal{C}_i = \langle \mu_i^{cpu}, \mu_i^{io} \rangle$, which encapsulates its verified computational throughput and storage I/O limits.

Threat Model: We assume a standard Byzantine fault-tolerant (BFT) bound, where the number of malicious validators f within any shard of size N_k is strictly constrained by $N_k \geqslant 3f + 1$. Within this security threshold, we consider two specific threats targeting heterogeneous sharding:

- *Hardware-induced State Divergence:* Honest nodes operating on disparate ISAs (e.g., x86 vs. ARM) inevitably produce microscopic floating-point deviations during GNN inference. Byzantine nodes can exploit these arithmetic inconsistencies to deliberately broadcast conflicting state roots, thereby fracturing the consensus.
- *Capacity Spoofing:* Malicious nodes may perform capacity spoofing by falsely reporting their hardware capabilities ($\mathcal{C}_i$) to evade heavy workloads, save computing resources, or maliciously induce the straggler effect. We explicitly bound our security assumptions to application-layer attacks. We assume the host operating system kernel remains uncompromised and correctly honors synchronous execution flags. Advanced attacks involving malicious operating system level kernel modifications, such as forcibly ignoring hardware I/O flags to cache ephemeral data in RAM, fall outside the scope of this protocol and require trusted execution environments or hardware enclaves to mitigate.

2.3 Problem Statement

Based on the aforementioned models, the objective of this work is to design a heterogeneity-native, deterministic sharding framework. Given a transaction graph $\mathcal{G}$ and a mixed cluster of hardware-profiled validators, the system aims to compute an optimal disjoint routing map $\mathcal{M} : A \to \{1, \ldots, K\}$ (where K is the number of shards) that simultaneously satisfies two rigorous constraints:

(1) **Load-Capacity Alignment (Performance):** The mapped transaction workload for each shard S_k must be strictly proportional to its aggregated physical capacity $\sum_{i \in S_k} \mathcal{C}_i$, thereby minimizing resource utilization variance and eliminating the straggler effect.

(2) **Cross-ISA Determinism (Safety):** The execution engine must guarantee that the calculation of $\mathcal{M}$ yields bit-perfect identical results across all honest nodes regardless of their underlying CPU/GPU architectures, completely nullifying the threat of state divergence.

Table 1 summarizes the key mathematical notations applied throughout this formulation.

Table 1 Summary of Notations

Symbol	Description
$\mathcal{V}, \mathcal{S}$	Set of validator nodes and set of shards
A, E	Set of accounts (nodes) and transactions (edges)
C_i	Verified processing capacity of node v_i
$\mathcal{G}$	Transaction interaction graph
Z	Unified Node Embedding Matrix ($Z \in \mathbb{Z}^{\lvert A \rvert \times d}$)
$w(u)$	Interaction frequency weight of account u
$\mathcal{M}$	Global sharding map for current epoch t
L_k, Cap_k	Workload and verified capacity of shard $\mathcal{S}_k$
$U_k, \bar{U}$	Shard utilization and global average utilization
λ	Hyperparameter balancing load fairness and locality
ϵ	Convergence threshold for rebalancing algorithm

3 The Proposed AdaH-Chain Framework

This section details the architecture and operational workflow of AdaH-Chain. As illustrated in Fig. 2, our framework addresses the heterogeneity challenge through three integrated layers: a secure capacity profiling layer, a deterministic execution engine, and a capacity-aligned rebalancing layer.

3.1 Architecture Overview

The AdaH-Chain workflow is triggered at the epoch boundary to reconfigure the state routing map. To safely and efficiently deploy learning-based heuristics across heterogeneous networks, our framework executes a strictly ordered three-phase pipeline. First, the Secure Capacity Discovery layer objectively quantifies the physical processing limits of all validators without trusting self-reported metrics. Second, the Deterministic GNN Execution Engine processes the transaction graph to extract topological embeddings; crucially, it does so without triggering arithmetic divergence across disparate ISAs.

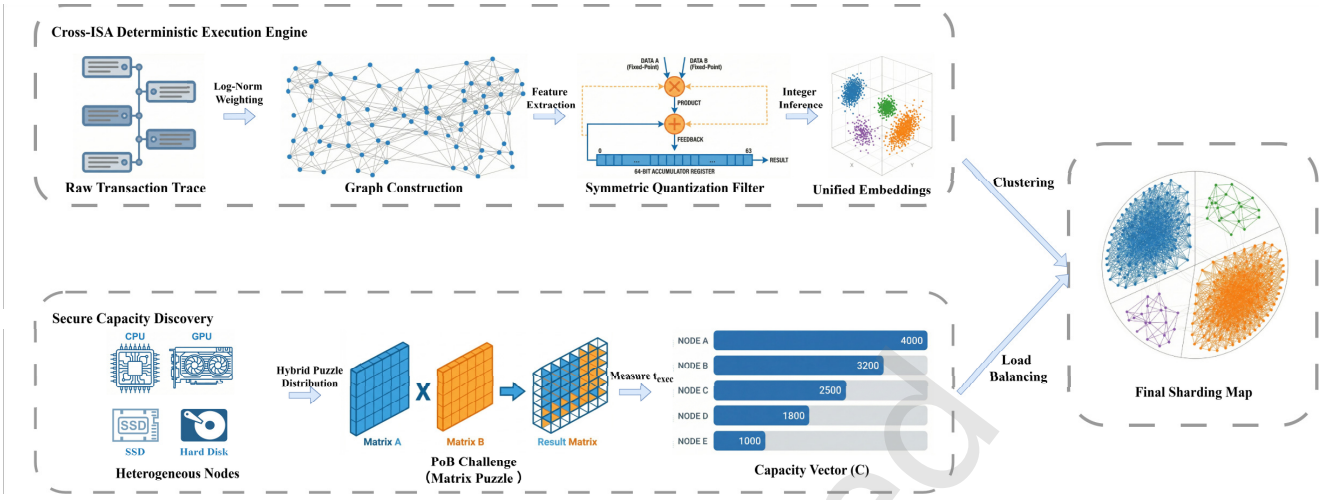


Fig. 2 The Architectural Workflow of AdaH-Chain. The Secure Capacity Discovery layer (Bottom) establishes a verified Capacity Vector $\mathcal{C}$ via PoB puzzles. The Cross-ISA Deterministic Execution Engine (Top) transforms raw transactions into consistent Node Embeddings Z through a quantization mechanism. These paths converge at the Capacity-Aware Sharding Layer (Right), where the rebalancing algorithm partitions the network into heterogeneous shards with balanced resource utilization.

Finally, the Capacity-Aware Rebalancing layer strictly aligns the extracted high-density transaction subgraphs with the verified hardware capacities. This pipeline ensures that workload distribution effectively eliminates the straggler effect while maintaining bit-perfect consensus safety.

3.2 Secure Capacity Discovery

To reliably quantify the processing power of untrustworthy nodes, we implement a cryptographic Proof-of-Benchmark (PoB) mechanism. This layer continuously profiles the mixed validator network by executing state-dependent storage reads combined with pseudo-random computational puzzles. By evaluating the execution latency within a standardized WebAssembly (Wasm) sandbox, the protocol generates a verifiable capacity vector $\mathcal{C}$.

This vector quantifies physical hardware limits, including I/O throughput and computational density, without exposing proprietary hardware details. The use of cryptographic commitments ensures that nodes cannot forge their capacity scores to manipulate workload assignment or save computing resources, thus providing a secure foundation for subsequent load balancing.

3.3 Deterministic GNN Execution Engine

Operating in parallel with hardware profiling, this engine executes a Graph Neural Network to extract logical routing topologies. The core challenge lies in the non-deterministic nature of the high-dimensional floating-point arithmetic inherent in standard GNN aggregations across different instruction sets (e.g., x86 AVX versus ARM NEON).

To enforce consensus safety without discarding the powerful topology extraction capabilities of GNNs, we strictly prohibit the use of hardware-native floating-point instructions. Instead, before graph aggregation, the engine passes incoming continuous interaction features through a symmetric quantization filter, mapping them into a discrete 32-bit integer domain (INT32). A purely integer-based (fixed-point) GNN inference phase then generates a Unified Node Embedding Matrix Z. By confining all neural network calculations to deterministic integer arithmetic, we guarantee that all validators arrive at a bit-perfect identical topological state, regardless of their underlying hardware architecture.

3.4 Capacity-Aware GNN Sharding

The final layer utilizes the verified capacity profile $\mathcal{C}$ and the deterministic embeddings Z to execute the capacity-aware rebalancing algorithm. Instead of enforcing the traditional equal shard size assumption, the protocol directs high-density transaction subgraphs, identified by the GNN embeddings, to high-capacity hardware nodes.

This mechanism minimizes the variance of resource utilization across the mixed cluster. By structurally aligning the computational intensity of transaction clusters with the physical throughput ceilings of the respective shards, AdaH-Chain prevents powerful nodes from idling while waiting for legacy validators. The resulting routing map $\mathcal{M}$ optimizes both intra-shard locality and global throughput, effectively preventing performance bottlenecks in heterogeneous environments.

4 Algorithmic Implementation and Mechanisms

Building upon the architectural overview, this section formally details the algorithmic execution of AdaH-Chain. We formulate the secure capacity discovery, the cross-ISA deterministic engine, and the capacity-aware sharding protocol.

To illustrate the operational mechanics throughout this section, we adopt a running scenario: consider a dense transaction subgraph $\mathcal{G}_{sub}$ containing thousands of interacting accounts within a heterogeneous financial network. The network state is partitioned into multiple consensus shards $\mathcal{S}_k \in \mathcal{S}$. We evaluate two distinct validators: a high-performance node v_1 (equipped with hardware accelerators and NVMe SSDs) and a commodity baseline validator v_3 (representing the minimum required performance) to derive their respective capacity scores C_1 and C_3.

4.1 Secure Capacity Discovery Implementation

As defined in our system model, each validator v_i possesses a multidimensional hardware profile $\mathcal{C}_i = \langle \mu_i^{cpu}, \mu_i^{io} \rangle$. To utilize this profile for sharding without trusting self-reported metrics, AdaH-Chain employs a State-Dependent Hybrid Proof-of-Benchmark (SD-PoB). This protocol simultaneously evaluates instruction-set throughput and bidirectional storage I/O limits (both disk read retrieval and disk write persistence), collapsing these dimensions into a single, verifiable integer metric $C_i \in \mathbb{N}^+$.

4.1.1 State-Dependent Puzzle Generation

To accurately quantify a node's true transaction processing capability, the SD-PoB forces the validator to resolve a strictly ordered cryptographic puzzle. Upon the transition to epoch t, the protocol derives a pseudo-random seed σ_t from the preceding block header H_{t-1} [15]:

$$\sigma_t = \text{Hash256}(H_{t-1} \parallel \text{EpochID}_t) \tag{2}$$

This seed σ_t initializes a sequence of D_{puz} hybrid execution steps. For each step $k \in [1, D_{puz}]$, the node must first retrieve historical state data Data_k from its local ledger database. This operation explicitly benchmarks the disk read bandwidth (μ_i^{io}):

$$\text{Data}_k = \text{StateRead}(\text{StateRoot}_{t-1}, \text{Path}_k(\sigma_t)) \tag{3}$$

where $\text{Path}_k(\sigma_t) = \text{Hash}(\sigma_t \parallel k)$ serves as a deterministic key derivation function. Consequently, the disk retrieval sequence is unpredictable to the validator but universally verifiable.

In parallel with the I/O task, the node executes a CPU-bound matrix multiplication $\mathbf{M}_k \times \mathbf{N}_k$ to benchmark its computational throughput (μ_i^{cpu}). Both matrices are deterministically generated using σ_t. To mathematically bind the CPU and I/O tasks, the protocol hashes both outputs into 256-bit scalars and fuses them using a bitwise XOR operator ($\oplus$):

$$R_k = \text{Hash256}(\mathbf{M}_k \times \mathbf{N}_k) \oplus \text{Hash256}(\text{Data}_k) \tag{4}$$

This XOR fusion acts as a strict cryptographic lock; a validator must complete both the compute and the retrieval tasks to derive the correct 256-bit intermediate state R_k.

In blockchain environments, state persistence (disk writes) often constitutes a more severe bottleneck than read operations. To benchmark the write capacity without polluting the globally finalized ledger, the protocol mandates the node to persist R_k into a local ephemeral sandbox (EphemeralDB_t) via synchronous I/O. This returns a boolean acknowledgment $\text{WriteAck}_k \in \{0, 1\}$:

$$\text{WriteAck}_k = \text{StateWrite}(\text{EphemeralDB}_t, \text{Path}_k(\sigma_t), R_k) \tag{5}$$

If $\text{WriteAck}_k = 0$, the intermediate result R_k is nullified. Finally, instead of simple addition, the protocol aggregates all valid intermediate states into a cumulative 256-bit cryptographic proof R_{final}:

$$R_{\text{final}} = \text{Hash256}\Big(\|_{k=1}^{D_{puz}} (R_k \cdot \text{WriteAck}_k) \Big) \tag{6}$$

where $\|$ denotes byte concatenation. The actual hardware capacity of the validator is objectively measured by the total temporal latency incurred while resolving this entire dependency chain.

4.1.2 Standardized Execution and Capacity Calculation

To prevent algorithmic biases from highly optimized proprietary libraries (e.g., hardware-specific math routines), AdaH-Chain mandates that the entire SD-PoB executes within a standardized WebAssembly (Wasm) Sandbox [16]. Both state retrieval and persistence must use synchronous direct I/O flags (e.g., `O_DIRECT`) to explicitly bypass kernel page caching [17].

Upon resolving the puzzle, each validator v_i broadcasts a cryptographic commitment:

$$\text{Commit}_i = \text{Sign}_{sk_i}(\text{Hash256}(R_{\text{final}} \parallel v_i \parallel \sigma_t)) \tag{7}$$

This signature binds the execution proof to the validator's identity. The network records the total execution latency $t_{\text{exec},i}$ taken by validator v_i to generate this valid commitment. To map this temporal metric into a discrete weight for sharding, the network computes the final capacity score C_i using an inversely proportional function:

$$C_i = \left\lfloor \frac{\kappa}{t_{\text{exec},i}} \right\rfloor \tag{8}$$

To ensure mathematical stability, the global scaling constant κ is anchored to the minimum hardware configuration permitted in the network. Let t_{base} denote the benchmarked execution latency of this baseline node, and C_{base} represent a configurable minimum integer score. The scaling constant is formally defined as $\kappa = t_{\text{base}} \times C_{\text{base}}$. To accommodate dynamic network churn and the potential departure of minimum-tier validators, the global scaling constant κ is dynamically recalibrated at the genesis of each epoch. The protocol queries the active validator set to identify the current lowest-performing node, updating t_{base} to ensure the ongoing validity of the integer mapping bounds. This guarantees that all compliant nodes ($t_{\text{exec},i} \leqslant t_{\text{base}}$) accurately map to a distinct integer capacity $C_i \geqslant C_{\text{base}}$.

Example 1 *We illustrate the SD-PoB calculation process using the high-performance node v_1 and the commodity baseline node v_3.*

First, the system initializes the scaling constant κ. Suppose the commodity node v_3 requires $t_{base} = 2.0$ seconds to complete the entire hybrid puzzle (including matrix multiplication, Wasm overhead, and synchronous disk I/O). Given a baseline capacity hyperparameter $C_{base} = 25$, the scaling constant evaluates to $\kappa = 2.0 \times 25 = 50$.

Next, the high-performance node v_1 executes the identical cryptographic puzzle. Benefiting from superior CPU clock speeds and NVMe SSD bandwidth, v_1 resolves the operations and broadcasts its valid commitment in a total latency of $t_{exec,1} = 0.5$ seconds.

Following Equation 8, the network computes their discrete capacities. For the baseline node v_3, the score evaluates to $C_3 = \lfloor 50/2.0 \rfloor = 25$. For v_1, the score is $C_1 = \lfloor 50/0.5 \rfloor = 100$. This mapping mathematically translates the verified temporal latency into an objective $4\times$ resource weight difference, entirely circumventing the need for trust.

4.2 Cross-ISA Deterministic Execution Engine

Before the capacity-aware sharding layer can balance workloads, it requires a deep structural understanding of the network topology. This is achieved through Graph Neural Network (GNN) inference, which outputs node embeddings. These embeddings mathematically encode dense interaction patterns into low-dimensional vectors, providing the precise numerical coordinates needed by the subsequent algorithm to cluster heavily communicating accounts into the same physical shard.

However, conventional GNN models depend heavily on hardware-accelerated floating-point arithmetic (FPA), which inherently conflicts with the bit-perfect determinism required by state machine replication. To resolve this, we introduce the Cross-ISA Deterministic Execution Engine. This layer entirely replaces continuous FPA with purely integer-based arithmetic, explicitly isolating the logical topology extraction from underlying hardware arithmetic variations.

4.2.1 *Graph Topology Construction*

AdaH-Chain models the global transaction state as a directed weighted graph $\mathcal{G} = (A, E)$. In real-world decentralized networks (e.g., Ethereum), interaction frequencies exhibit a severe heavy-tailed distribution; interactions with DeFi routers can reach tens of millions, while regular accounts have few. Passing raw, unbounded transaction counts into a GNN causes gradient instability and heavily skews the message-passing mechanism.

Traditional data science resolves this using logarithmic functions (e.g., $\ln(x)$). However, computing logarithms requires non-deterministic floating-point math libraries (e.g., `libm`). To achieve hardware-agnostic compression, the protocol evaluates the directed edge weight w_{uj} using a native integer bit-length operator:

$$w_{uj} = \text{BitLen}(1 + \text{Count}_{u \to j}) \tag{9}$$

where $\text{Count}_{u \to j}$ denotes the historical interactions from source $u \in A$ to destination $j \in A$. The $\text{BitLen}(x)$ operator derives the position of the most significant bit (MSB), essentially functioning as an integer approximation of $\log_2(x)$. This operator executes as a single-cycle deterministic integer instruction across all ISAs (e.g., `BSR` in x86, `CLZ` in ARM). This explicitly compresses the heavy-tailed transaction count into a compact discrete integer ($w_{uj} \in \mathbb{N}$), ensuring the graph topology is intrinsically float-free. While logarithmic binning inherently drops microscopic frequency variance, this precision loss is a controlled engineering trade-off that does not degrade macro-level partitioning. Preserving the exact numerical difference between 1,000,000 and 1,000,050 interactions provides no topological utility for community detection. The BitLen operator successfully retains the magnitude difference separating central hub contracts from regular accounts, providing sufficient structural resolution for the algorithm. The topology is persisted using the Compressed Sparse Row (CSR) format [18].

4.2.2 *Symmetric Quantization Mechanism and Fixed-Point GNN Inference*

To execute the GNN, the system must perform Multiply-Accumulate (MAC) operations involving both the integer edge weights and the node features. If node features remain as floating-point tensors, the entire computation is tainted by

cross-ISA rounding deviations (e.g., varying fused multiply-add execution orders between CPU and GPU architectures). To eliminate this state divergence hazard, we propose a Symmetric Quantization Filter that maps continuous node features strictly into the 32-bit signed integer (INT32) domain prior to any GNN aggregation.

The filter first establishes a robust feature boundary using the Interquartile Range (IQR) [19]. Let $P_{99.9}$ and Q_3 denote the 99.9th percentile and the 75th percentile of the absolute node feature values $|X_{t-1}|$ from the preceding epoch. We define a safe maximum threshold θ_{safe} to filter extreme anomalies:

$$\theta_{safe} = \min(P_{99.9}, Q_3 + 1.5 \times \text{IQR}) \tag{10}$$

The selection of these specific statistical boundaries is mathematically grounded in the scale-free nature of decentralized networks. Relying exclusively on $P_{99.9}$ exposes the mapping scale to distortion by a single extreme outlier, whereas relying solely on the IQR risks aggressively clipping the latent features of highly connected core accounts. By adopting the minimum of these two bounds, the system guarantees that the feature representations of critical network hubs remain intact while preventing isolated anomalies from compressing the effective resolution of the entire quantization grid. To guarantee that all heterogeneous validators apply the exact same quantization logic, the epoch leader deterministically computes the global scaling factor α_t and embeds it within the block header:

$$\alpha_t = \frac{2^{31} - 1}{\theta_{safe} + \beta} \tag{11}$$

where $2^{31} - 1$ is the physical INT32 upper bound, and β prevents division by zero. All validators then project the continuous feature matrix X_t into the integer domain using the uniform quantization function $Q(X_t)$:

$$X_{int} = Q(X_t) = \text{clip}\left(\lfloor X_t \cdot \alpha_t \rceil, -2^{31}, 2^{31} - 1\right) \tag{12}$$

where $\lfloor \cdot \rceil$ denotes nearest-integer rounding, and $\text{clip}(\cdot)$ strictly enforces the INT32 physical memory boundaries.

Algorithm 1 details the deterministic generation procedure of the Unified Node Embedding Matrix $Z \in \mathbb{Z}^{|A| \times d}$, leveraging the quantized integers to completely eliminate floating-point states during GNN message passing.

As depicted in Algorithm 1, to prevent non-determinism caused by OS-level thread scheduling and unordered hash-map iterators, we process each account u in strict ascending order of its ID (Line 2). This guarantees an identical mathematical execution trace across all nodes.

Crucially, the algorithm maps cleanly to hardware registers. The maximum quantized feature is bounded by $2^{31} - 1$. The discrete edge weight w_{uj}, even for a massive smart contract with 1 billion interactions, evaluates to $\text{BitLen}(10^9) \approx 30$. Thus, a single integer multiplication ($X_{int} \times W_{edge}$) peaks at $\approx 6.4 \times 10^{10}$, perfectly avoiding overflow in a standard 64-bit CPU accumulator (Acc). To constrain the data width for subsequent neural network layers, the 64-bit sum is deterministically bit-shifted ($\gg$) and clamped back into the 32-bit INT32 domain (Lines 12-13).

Algorithm 1 Deterministic Fixed-Point GNN Forward Pass

Input: Continuous Node Features X_t, Adjacency Matrix $\mathbf{Adj}$, Global Scaling Factor α_t
Output: Integer Embeddings $Z \in \mathbb{Z}^{|A| \times d}$

1: $X_{int} \leftarrow Q(X_t)$ *// Map to INT32 using Eq. 12*
2: **for** each account $u \in A$ in strict ascending order of ID **do**
3: $Acc \leftarrow \mathbf{0}^d$ *// Initialize 64-bit integer accumulator*
4: $\mathcal{N}(u) \leftarrow \text{GetNeighbors}(u, \mathbf{Adj})$
5: **if** $|\mathcal{N}(u)| > N_{max}$ **then**
6: $\mathcal{N}(u) \leftarrow \text{Top\_}N_{max}(\mathcal{N}(u), \text{sorted by } w_{uj})$
7: **end if**
8: **for** each neighbor $j \in \mathcal{N}(u)$ **do**
9: $W_{edge} \leftarrow w_{uj}$
10: $Acc \leftarrow Acc + (X_{int}[j] \times W_{edge})$ *// Native 64-bit MAC*
11: **end for**
12: $\texttt{SHIFT} \leftarrow \text{BitLen}(\max(|\mathcal{N}(u)|) \cdot \lfloor \alpha_t \rfloor)$
13: $Z[u] \leftarrow \text{clip}(Acc \gg \texttt{SHIFT}, -2^{31}, 2^{31} - 1)$ *// Downcast to INT32*
14: **end for**
15: **return** Z

Example 2 *We trace a single logical edge (a_1, a_2) to demonstrate how the execution engine physically prevents consensus collapse. Suppose the network analyzes the preceding epoch and derives a robust boundary $\theta_{safe} = 150325.0$. According to Eq. 11, the global scaling factor is evaluated as $\alpha_t = (2^{31} - 1)/150325.0 \approx 14285.54$.*

Due to architectural differences in floating-point FMA units, the continuous raw feature is evaluated microscopically differently across the network: it yields 0.85320001 on an x86 validator, but 0.85319999 on an ARM validator. If used directly, this 0.00000002 deviation would irreversibly fork the ledger state root.

However, before entering the GNN, the quantization filter $Q(\cdot)$ rigorously maps these floats into the discrete domain.

For the x86 node:

$$X_{int,x86} = clip(\lfloor 0.85320001 \times 14285.54 \rceil) = \lfloor 12188.423 \rceil = 12188$$

Simultaneously, for the ARM node:

$$X_{int,ARM} = clip(\lfloor 0.85319999 \times 14285.54 \rceil) = \lfloor 12188.422 \rceil = 12188$$

*The symmetric quantization filter absorbs the hardware-induced arithmetic noise. Consequently, an identical integer feature (*12188*) is passed into the graph aggregator, mathematically guaranteeing absolute bit-level consensus across all ISAs.*

4.3 Capacity-Aware GNN Sharding

With the deterministic embeddings Z generated in the previous layer, accounts with dense interactions are mapped to similar numerical coordinates. The final layer of AdaH-Chain leverages these coordinates to partition the global state. Our objective is to balance the transaction workload proportionally to the physical hardware capacity of each shard, while keeping tightly connected accounts together to minimize cross-shard transactions.

4.3.1 Sharding Objective Formulation

We formalize this goal through a dual-objective optimization function:

$$\min_{\mathcal{M}} J = \lambda \cdot \underbrace{\frac{1}{K}\sum_{k=1}^{K}\left(\frac{U_k - \bar{U}}{\bar{U}}\right)^2}_{\text{Capacity Utilization Variance}} + (1-\lambda)\cdot \underbrace{\frac{1}{|E|}\sum_{(u,j)\in E}\mathbb{I}(\mathcal{M}(u)\neq\mathcal{M}(j))}_{\text{Edge-Cut Ratio}} \tag{13}$$

In Equation 13, $\mathcal{M} : A \to \{1, \ldots, K\}$ represents the global sharding map. The parameter $\lambda \in [0, 1]$ is a system hyperparameter that explicitly balances workload fairness against cross-shard communication overhead. The second term penalizes the edge-cut ratio: the indicator function $\mathbb{I}(\cdot)$ evaluates to 1 if an edge spans across two disparate shards (constituting a cross-shard transaction), and 0 otherwise.

To evaluate the capacity utilization defined in the first term, we formalize the metrics of transaction workload and physical capacity. We define the integer workload $w(u)$ of an account u as its total transaction count. Thus, the total assigned workload for shard k is $L_k = \sum_{\mathcal{M}(u)=k} w(u)$. Using the verified hardware profile derived in Section 4.1, the total processing capacity of shard k is $Cap_k = \sum_{v_i \in \mathcal{V}_k} C_i$. The utilization rate of shard k is explicitly defined as $U_k = L_k / Cap_k$.

Minimizing this variance directly requires continuous floating-point differentiation, which is prohibited in BFT environments. To preserve strict consensus safety, AdaH-Chain translates this continuous variance problem into a deterministic integer-target matching procedure.

4.3.2 Deterministic Rebalancing Algorithm

Algorithm 2 presents a deterministic heuristic to compute the optimized sharding map using strictly integer operations.

Algorithm 2 Capacity-Aware Integer Rebalancing Strategy

Input: Embeddings Z, Shard Capacities $\{Cap_k\}_{k=1}^{K}$, Account Workloads $\{w(u)\}_{u\in A}$
Output: Deterministic Sharding Map $\mathcal{M}$

1: $\mathcal{M} \leftarrow$ EpochLeader_Propose_KMeans(Z, K)
2: Compute totals W_{total}, C_{total}, and initialize shard workloads L_k
3: **for** $k = 1$ **to** K **do**
4: $\quad Target_k \leftarrow \lceil (W_{total} \cdot Cap_k)/C_{total} \rceil$
5: **end for**
6: **for** $t = 1$ **to** T **do**
7: $\quad S_{src} \leftarrow \arg\max_k(L_k - Target_k)$; $S_{dst} \leftarrow \arg\min_k(L_k - Target_k)$
8: $\quad$ **if** $(L_{src} - Target_{src}) - (L_{dst} - Target_{dst}) \leqslant \epsilon$ **then break**
9: $\quad \mathcal{B} \leftarrow$ GetBoundaryNodes(S_{src}, S_{dst})
10: $\quad$ **if** $\mathcal{B} = \emptyset$ **then** $S_{dst} \leftarrow$ FallbackReroute(S_{dst}); **continue**
11: $\quad$ **for** $u \in$ SortAscending$(\mathcal{B}, w)$ **subject to** $L_{dst} + w(u) \leqslant Target_{dst}$ **do**
12: $\quad\quad \mathcal{M}[u] \leftarrow S_{dst}$; $L_{dst} \leftarrow L_{dst} + w(u)$; $L_{src} \leftarrow L_{src} - w(u)$
13: $\quad$ **end for**
14: **end for**
15: **return** $\mathcal{M}$

The process relies on the "Compute Off-chain, Verify On-chain" paradigm. In the initialization phase (Line 1), the current Epoch Leader locally computes a baseline map $\mathcal{M}$ by applying K-Means clustering to the embeddings Z. Since K-Means involves non-deterministic floating-point centroids, other validators do not re-execute it. Instead, they receive the leader's map $\mathcal{M}$ and deterministically verify its validity using integer accumulation. If the initial edge-cut penalty computed by the validators exceeds a predefined system tolerance threshold τ, the proposal is rejected, triggering a PBFT View-Change to replace the leader.

Once verified, the protocol computes a strict integer workload target $Target_k$ for each shard (Lines 3-4). We use scaled division combined with a ceiling operator ($\lceil \cdot \rceil$). This operator guarantees that $\sum Target_k \geqslant W_{total}$, ensuring that all transaction workloads can be fully allocated without leaving "orphan accounts" due to integer truncation.

Within the migration loop (Lines 6-14), the algorithm dynamically identifies the most overloaded shard S_{src} and the most underloaded shard S_{dst}. It then extracts the topological boundary set $\mathcal{B}$, which consists of accounts located in S_{src} that possess direct transaction edges to S_{dst}. The loop terminates based on a dynamic convergence threshold ϵ. Rather than applying a static scalar that could risk premature halting, ϵ is mathematically initialized as $\max(w_{avg}, 0.05 \times Target_{dst})$, where w_{avg} is the average transaction weight of the accounts in $\mathcal{B}$. During our empirical simulations, this threshold consistently converged near 150 transactions, ensuring the execution safely terminates once the remaining load variance aligns with the natural discrete granularity of the account weights. Migrating these specific accounts relieves the overloaded shard without creating new cross-shard overhead. To further optimize the edge-cut ratio, accounts in $\mathcal{B}$ are sorted and migrated in ascending order of their weights until the underloaded shard hits its capacity ceiling ($Target_{dst}$).

Example 3 *To demonstrate the rebalancing logic, suppose the network contains a high-performance Shard $\mathcal{S}_1$ (Hardware Score $Cap_1 = 400$) and a resource-constrained Shard $\mathcal{S}_3$ (Hardware Score $Cap_3 = 100$). The total physical capacity is $C_{total} = 500$.*

Assume the total transaction workload to be processed is $W_{total} = 600$ transactions. The initial clustering assigns $L_1 = 300$ and $L_3 = 300$ transactions. The utilization rates are heavily skewed: $U_1 = 300/400 = 0.75$, while $U_3 = 300/100 = 3.0$ (severely overloaded).

Instead of computing floating-point variances, the protocol calculates proportional integer transaction targets: $\mathcal{S}_1$ should handle $Target_1 = \lceil 600 \times 400/500 \rceil = 480$ transactions, and $Target_3 = 120$ transactions.

During migration, the algorithm extracts the boundary set $\mathcal{B}$ between the overloaded $\mathcal{S}_3$ and the underloaded $\mathcal{S}_1$. Suppose $\mathcal{B}$ contains a lightweight account a_3 ($w(a_3) = 150$ txs) and a heavy account a_4 ($w(a_4) = 250$ txs). Algorithm 2 prioritizes a_3. Since moving a_3 satisfies the strict capacity ceiling of $\mathcal{S}_1$ ($L_1 + 150 = 450 \leqslant 480$), the migration safely executes. Post-migration, the workloads shift to $L_1 = 450$ and $L_3 = 150$, resulting in optimized and nearly identical utilization rates ($U_1 = 1.125$, $U_3 = 1.5$). This effectively eliminates the straggler effect using pure integer logic.

4.4 System Optimization and Complexity

To prove the practical feasibility of AdaH-Chain in high-throughput environments, this section introduces two system-level engineering optimizations: parallel graph inference and asynchronous state synchronization. It concludes with a rigorous theoretical complexity analysis.

4.4.1 *Parallelized Graph Inference*

In Algorithm 1, the embedding computation for different accounts is strictly independent. We exploit this data-level parallelism by dispatching disjoint subsets of accounts to a multi-core thread pool. Because financial transaction graphs inherently exhibit power-law degree distributions, uniformly dispatching accounts inevitably creates severe synchronization bottlenecks. To structurally balance the intra-node workload prior to the global synchronization barrier, the multi-core thread pool utilizes dynamic work-stealing. Accounts are sorted sequentially by their topological degree and partitioned into discrete processing blocks. Idle physical CPU cores dynamically fetch unprocessed blocks from the shared queue, mathematically guaranteeing that processing threads handling dense subgraphs remain load-balanced against those assigned to sparse regions. This shared-memory parallelization reduces the dominant GNN inference time complexity from $\mathcal{O}(|E| \cdot d)$ to $\mathcal{O}(|E| \cdot d/P + \log P)$, where P is the number of physical CPU cores and d is the embedding dimension.

4.4.2 *Asynchronous State Synchronization*

Computing a new sharding topology for millions of accounts takes several seconds. To prevent consensus stalling, this computation is decoupled from the critical path and executed as a background process.

Furthermore, migrating accounts across shards involves heavy state transfers. To avoid halting the network, we implement Background State Prefetching. The target shards prefetch the historical state of migrating accounts asynchronously, δ blocks prior to the epoch boundary. During this narrow migration window, the migrating accounts are placed under a temporary Read-Only Lock. Any incoming write transactions targeting them are temporarily buffered in the target shard's mempool. At the epoch boundary, the locks are released, and the buffered transactions are executed sequentially.

This design guarantees linearizability without sacrificing liveness. For example, assuming a worst-case 15-second computation window at a peak throughput of 4,800 TPS, the network generates 72,000 intermediate transactions. Given that a standard Ethereum transaction payload consumes approximately 250 bytes, buffering these transactions imposes a peak memory footprint of roughly 18 MB. This nominal allocation

is comfortably absorbed by local RAM. To fundamentally prevent Out-of-Memory (OOM) failures on resource-constrained validators during extreme network anomalies, the mempool implements deterministic disk-spilling. If the buffered queue surpasses a predefined 50 MB hardware safety threshold, the system asynchronously flushes the overflow payload to local NVMe storage, ensuring uninterrupted liveness. The corresponding state trie updates typically compress to less than 15 MB. Transmitting this delta state across a standard 1 Gbps intra-cluster network requires only ~ 150 ms, which comfortably fits within a standard 500 ms PBFT block interval without triggering timeout views.

4.4.3 *Complexity and Overhead Analysis*

The system overhead is strictly bounded by its two heaviest operations. Using the CSR format restricts the spatial and temporal complexity of the fixed-point GNN inference to $\mathcal{O}(|E| \cdot d)$. The capacity rebalancing phase relies exclusively on integer sorting of boundary nodes, which operates at $\mathcal{O}(|A| \log |A|)$ in the worst case.

Our empirical benchmarks on a standard 8-core commercial CPU confirm that a full topological reconfiguration for one million accounts completes in approximately 13.4 seconds. Within a standard 24-hour epoch ($86,400$ seconds), this reconfiguration represents a negligible 0.015% temporal overhead. This confirms that AdaH-Chain is structurally viable for large-scale, high-throughput production environments.

5 Theoretical Analysis

This section formally analyzes the safety, convergence, and security properties of the AdaH-Chain framework, demonstrating how the system-level mechanisms established in Section 4 mathematically guarantee deterministic consensus and fault tolerance.

5.1 Consensus Safety Analysis

In sharded ledgers, all honest validators within a consensus group must maintain an identical state machine. Hardware-induced arithmetic divergence breaks this fundamental requirement. For instance, if a heterogeneous shard consists of CPU and GPU nodes, native floating-point rounding disparities will yield conflicting state roots, triggering a permanent PBFT liveness deadlock.

Definition 1: (Cross-ISA Consistency). *A sharding execution protocol is consistent if and only if, given an identical input state S, the computation yields an identical output topology map $\mathcal{M}$ across all heterogeneous hardware profiles $\mathcal{H}$:* $\forall h_i, h_j \in \mathcal{H}, Exec_{h_i}(S) = Exec_{h_j}(S)$.

Theorem 1: (Cross-ISA Consistency in GNN Inference). The fixed-point execution engine mathematically guarantees Cross-ISA Consistency.

Proof: The consistency is established through three explicit hardware-agnostic constraints. First, the quantization mapping (Equation 12) coerces all continuous inputs into the discrete INT32 domain, completely eliminating floating-point operands from the feature space. Second, the algorithm enforces a single-threaded logical execution sequence by sorting the neighbor lists by Account IDs (Algorithm 1), ensuring that the memory access and accumulation trace is unique regardless of OS-level thread scheduling. Finally, during message passing, the integer Multiply-Accumulate operations are strictly bounded within 64-bit accumulators without triggering physical register overflow. Because INT64 addition and multiplication are strictly deterministic across all modern ISAs, processing any graph G yields a bit-perfect identical embedding matrix Z. □

5.2 Rebalancing Convergence Analysis

To guarantee global liveness, the dynamic topology reconfiguration phase must execute deterministically and terminate within a bounded temporal window.

Theorem 2: (Convergence Bound). The integer rebalancing strategy mathematically terminates in finite steps, bounded by $\mathcal{O}(\frac{\Phi_0}{w_{min}})$, where Φ_0 is the initial absolute load error and w_{min} is the minimum account interaction weight.

Proof: Let $\Phi = \sum_{k=1}^{K} |L_k - Target_k|$ denote the total absolute load error of the system. In each migration iteration, the protocol selects an account u with weight $w(u)$ and moves it from an overloaded shard S_{src} (where $L_{src} > Target_{src}$) to an underloaded shard S_{dst} (where $L_{dst} < Target_{dst}$). The algorithm explicitly enforces the destination capacity ceiling: $L_{dst} + w(u) \leqslant Target_{dst}$. Consequently, this specific migration strictly reduces the absolute error of S_{dst} by exactly $w(u)$. Concurrently, the migration reduces the load of S_{src}. Even if S_{src} transitions from an overloaded to an underloaded state during this single step, the aggregate system error Φ is mathematically guaranteed to decrease by at least $w(u)$. Given that the discrete transaction count dictates $w_{min} \geqslant 1$, the total error Φ decreases monotonically by a positive integer bound in each valid iteration. Since the initial error Φ_0 is finite, the algorithm must terminate in $\mathcal{O}(\frac{\Phi_0}{w_{min}})$ steps, effectively avoiding infinite loops. □

5.3 Security of Capacity Discovery

Theorem 3: (Resistance to Capacity Spoofing). Assuming the cryptographic hash function is collision-resistant, a mali-

cious node with physical capacity C_{real} cannot forge a valid execution proof for a claimed capacity $C_{claimed} > C_{real}$.

Proof: The SD-PoB puzzle generation strictly couples CPU computation and disk I/O. The bitwise XOR fusion operator ($R_k = \text{Hash256}(\mathbf{M}_k \times \mathbf{N}_k) \oplus \text{Hash256}(\text{Data}_k)$) acts as a cryptographic lock, ensuring that deriving the correct intermediate state requires the precise execution of both hardware domains. A Byzantine node cannot bypass the I/O state retrieval, because missing historical data completely alters the final cumulative hash R_{final}. Since the initial seed σ_t is unpredictable, pre-computation is structurally impossible. Thus, the elapsed latency t_{exec} serves as a mathematically unforgeable lower bound of the node's true throughput. □

System Remark: (Mitigating Bait-and-Switch Attacks). While Theorem 3 prevents algorithmic forgery, a commodity node might temporarily rent high-performance cloud instances during the SD-PoB window to fraudulently secure a high capacity score. To neutralize this attack, AdaH-Chain couples the profiling with runtime slashing. To explicitly differentiate deliberate capacity spoofing from benign transient network congestion or temporary hardware throttling, the protocol relies on historical latency variance. The network evaluates node performance over a sliding window spanning multiple consecutive consensus rounds. A slashing condition is triggered exclusively when a validator's latency consistently deviates from its claimed tier limits beyond a statistically calculated jitter tolerance boundary. This statistical isolation ensures that honest nodes remain unaffected by uncontrollable network noise, while sustained malicious outsourcing receives permanent capacity downgrades and economic penalties.

5.4 Fault Tolerance against Malicious Leaders

Beyond standard intra-shard PBFT failures, the architecture must secure the network against Byzantine Epoch Leaders during the topology reconfiguration phase.

A malicious leader could execute a Topology Degradation Attack by proposing a valid sharding map $\mathcal{M}$ that satisfies all integer capacity constraints but deliberately maps tightly coupled accounts into disjoint shards. This adversarial behavior aims to artificially inflate the cross-shard communication overhead and paralyze the network throughput.

To neutralize this threat, the framework enforces an On-chain Validity Bound. Because the verification paradigm ensures all validators independently possess the deterministic embeddings Z and the proposed map $\mathcal{M}$, every validator locally evaluates the dual-objective function (Equation 13). If the resultant edge-cut ratio exceeds a predefined system tolerance threshold τ, the validators deterministically reject the proposal. This initiates a PBFT View-Change to replace the Byzantine leader, ensuring that the structural reconfiguration strictly optimizes the global system performance.

6 Evaluation

6.1 Experimental Setup

We conduct experiments to evaluate the performance and robustness of AdaH-Chain. We utilize a trace-driven simulation that combines physical hardware profiling with distributed network emulation. First, we deploy the fixed-point mechanism and graph algorithms on physical testbeds to measure baseline execution limits. Then, we integrate the generated deterministic sharding maps and hardware constraints into the *BlockEmulator* [20] framework to measure distributed metrics, including global throughput and cross-shard latency.

For the workloads, we extract a real-world dataset containing 1,000,000 historical transactions from the Ethereum Mainnet. To prevent out-of-memory (OOM) failures when processing this large-scale graph, we apply the CSR storage optimization for AdaH-Chain. To rigorously evaluate the clustering overhead at scale, we configure the emulator with a fixed shard-to-node ratio where each logical shard consistently comprises 4 physical validator nodes. Across our experimental configurations, the system scales from a minimum of 4 shards (16 total nodes) up to a maximum of 16 shards (64 total nodes) to collectively manage the 1,000,000 historical accounts. This setup accurately reflects the structural scaling of heterogeneous networks while isolating the exact computational and memory overhead borne by the validators when processing dense transaction subgraphs.

6.1.1 *Physical Testbed and Heterogeneity Configuration*

To extract accurate execution limits and validate the Cross-ISA deterministic engine, we deploy microbenchmarks across three distinct architectures: (1) a consumer laptop (AMD Ryzen 7, 24GB RAM, x86-64); (2) an enterprise workstation (Intel Core i9, NVMe SSD, x86-64); and (3) a cloud instance (Intel Xeon, NVIDIA T4 GPU, CUDA).

We parameterize this heterogeneity into a two-tier abstraction for large-scale network simulations, implementing a $4\times$ performance gap between high-performance nodes ($C_{strong} = 4.0$, 500ms block interval) and resource-constrained baseline nodes ($C_{weak} = 1.0$, 2000ms block interval).

6.1.2 *Baselines*

To systematically evaluate AdaH-Chain, we categorize existing sharding protocols into three paradigms:

The first paradigm is the Topology-Oblivious mechanism, represented by Random Sharding (OmniLedger [12]). This protocol assigns accounts via cryptographic hashing, serving as the baseline for uniform, unoptimized routing.

The second paradigm focuses on State-Decision Learning, represented by modern Deep Reinforcement Learning (DRL) architectures such as SPRING [1] and AERO [2]. These protocols frame state partitioning as a Markov Decision Process to dynamically schedule workloads. To measure their theoretical maximum throughput without triggering state divergence, we isolate their floating-point inference to an off-chain oracle.

The third paradigm centers on Graph Representation, where we evaluate two distinct configurations: Native GNN and GNN (INT32). Native GNN utilizes hardware floating-point units to process heuristics. We use it to measure the actual consensus divergence rate across mixed CPU-GPU environments. Conversely, GNN (INT32) serves as an ablation baseline; it utilizes our deterministic engine but strictly enforces uniform load distribution. This isolates the performance gains contributed specifically by our capacity-aware rebalancing logic.

6.2 Evaluation Methodology

To validate AdaH-Chain against the challenges identified in Section 1, we design four sets of experiments. First, we evaluate the SD-PoB overhead (Section 6.3) and the Cross-ISA Engine (Section 6.4) to address the profiling and consistency challenges. Second, we demonstrate how Capacity-Aware Sharding resolves the straggler effect (Section 6.5). Finally, we conduct a robustness analysis under extreme network conditions (Section 6.7).

6.3 Overhead of Capacity Discovery

Settings: To evaluate the execution overhead of the State-Dependent Proof-of-Benchmark (SD-PoB), we execute cryptographic puzzles ($D_{puz} = 50$ with 512×512 matrix dimensions) across the three physical testbeds. Fine-grained timers decouple the execution latency into I/O StateRead operations and CPU-intensive matrix accumulations.

Results: Table 2 reports the decoupled latency metrics and the derived capacity scores. The execution overhead is negligible (< 0.5 seconds) across all devices, confirming that SD-PoB avoids introducing computational bottlenecks at the epoch boundary. Furthermore, the decoupled metrics reveal architectural mismatches invisible to traditional sharding protocols. The enterprise workstation exhibits high storage I/O performance ($C_{io} = 57876$) due to its NVMe SSD. However, relying solely on its CPU for Wasm-based dense matrix operations yields a lower computational score ($C_{cpu} = 209$) compared to the massively parallel GPU cloud instance ($C_{cpu} = 8171$). Conversely, the cloud instance falls behind the workstation in raw I/O access.

Summary: If a performance-agnostic model blindly assigns a computationally heavy transaction subgraph to the workstation, it creates a straggler bottleneck. Our composite capacity profile C_{total} natively captures this multi-dimensional constraint, proving that relying purely on single-dimensional metrics is inadequate.

Table 2 Fine-Grained Latency and Derived Scores of SD-PoB

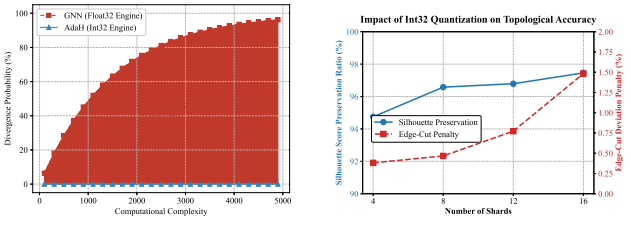
Hardware	I/O(s)	CPU(s)	Total	C_{io}	C_{cpu}	C_{total}
Laptop (AMD)	0.0218	0.2318	0.2536	4577	431	394
Workstation (Intel)	0.0017	0.4778	0.4795	57876	**209**	**208**
Cloud (T4 GPU)	0.0023	0.0122	0.0145	43672	8171	6883

6.4 Impact of Deterministic Execution

Settings: To validate the execution engine in preventing consensus divergence, we simulate GNN aggregation on mixed x86/CUDA clusters and compare the Float32 divergence rate against our Int32 precision loss.

Results: Figure 3(a) illustrates that the standard Float32 arithmetic of the Native GNN causes a consensus divergence rate that quickly reaches 100%. In contrast, our Int32 mechanism maintains a strict 0% divergence rate. Figure 3(b) illustrates that the quantization introduces less than a 1.5% penalty to the final edge-cut ratio, preserving 94.7% of the original Silhouette Score [21].

Summary: AdaH-Chain effectively resolves the hardware-induced floating-point divergence while trading only a minimal topological accuracy penalty for strict cross-platform consensus safety.



(a) Consensus Divergence Probability

(b) Topological Accuracy Trade-off

Fig. 3 Cross-ISA Determinism vs. Accuracy Trade-off. The fixed-point engine preserves 94.7% of the continuous silhouette score, introducing less than a 1.5% penalty to the final edge-cut ratio while ensuring complete consensus safety.

6.5 Overall Performance

Settings: We measure global scalability by varying the active shard count from 4 to 16. We also analyze the Cumulative Distribution Function (CDF) of transaction confirmation latency under heavy network loads. Native GNN is excluded from steady-state evaluations because its non-deterministic state execution prevents the maintenance of a functional consensus, rendering steady-state block production impossible. The comparison focuses on Random, DRL, and GNN (INT32).

Results: Figure 4(a) indicates that AdaH-Chain achieves a peak throughput of 4,800 TPS under the 16-shard configuration. This delivers a $4.8\times$ improvement over the Random Sharding baseline ($\approx 1,000$ TPS), and a $1.9\times$ gain over the DRL and GNN (INT32) baselines (which saturate near 2,500 TPS). Concurrently, Figure 5 demonstrates that the baselines suffer from long-tail latency (P99 $>$ 30 seconds) due to straggler queues on resource-constrained nodes. AdaH-Chain bounds the P99 latency within 5 seconds.

Summary: Aligning transaction workloads with physical processing ceilings eliminates the straggler effect, ensuring high global throughput and predictable service latency.

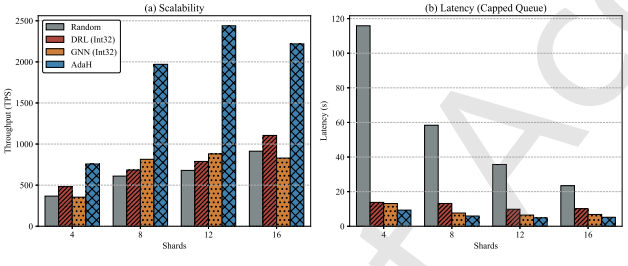


Fig. 4 Performance comparison. (a) Throughput scalability across multiple shards; (b) Average transaction confirmation latency.

6.6 Evaluation of Load Balancing

Settings: To understand the source of the performance improvements, we evaluate the workload distribution and the Cross-Shard Transaction (CST) ratio.

Results: Figure 6 highlights the block production variance. The DRL baseline enforces a strict 20% CST ratio, which causes severe workload imbalance: Shard 0 processes 31 blocks while Shard 1 processes only 9. In contrast, AdaH-Chain allows a higher CST ratio (45%) to dynamically migrate boundary nodes, precisely matching the block production rate with the hardware tiers. Figure 7 further includes the GNN (INT32) variant. While GNN (INT32) minimizes the CST ratio, its throughput saturates near 2,500 TPS (Figure 4(a)). This ceiling occurs because uniform graph partitioning forces high-performance nodes to idle while waiting for legacy

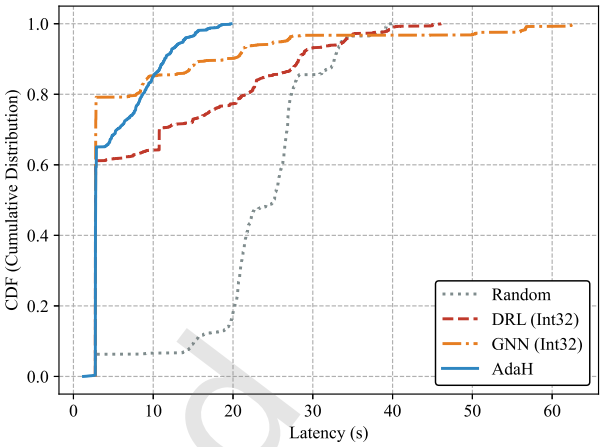


Fig. 5 Cumulative Distribution Function (CDF) of transaction latency. AdaH-Chain bounds the 99th percentile (P99) latency effectively by eliminating localized stragglers.

validators. Furthermore, the selection of the dual-objective hyperparameter λ is grounded in a boundary analysis of the system's operational extremes. Mathematically, setting $\lambda = 1.0$ completely eliminates the straggler effect by strictly prioritizing hardware capacity alignment, but it artificially inflates the cross-shard transaction ratio to nearly 60%, congesting the network with message relays. Conversely, setting $\lambda = 0.0$ aggressively preserves graph locality (restricting the cross-shard penalty to below 15%), but causes severe workload imbalance that forces high-performance nodes into prolonged idle states. By sampling these boundary conditions during the network initialization phase, we calibrated λ to 0.65. This configuration provides the optimal operational trade-off, maximizing global throughput without triggering communication bottlenecks.

Summary: Trading a moderate cross-shard communication penalty for strictly aligned hardware utilization is highly effective. Preserving graph locality provides no throughput benefits if the assigned local hardware lacks the capacity to process the dense workload.

6.7 Robustness Analysis

Settings: We stress-test the protocol under extreme network conditions. We scale the hardware performance gap from $1\times$ to $8\times$ (Fig. 8) and vary the proportion of strong nodes from 12.5% to 50% (Fig. 9). We inject cross-shard network delays of up to 5,000 ms (Fig. 10) and measure the epoch reconfiguration time for topologies scaling up to 1,000,000 accounts (Fig. 11).

Results: Figure 8 shows that when the heterogeneity gap reaches $8\times$, AdaH-Chain maintains a peak throughput above 1,000 TPS, whereas the DRL baseline drops below 600 TPS

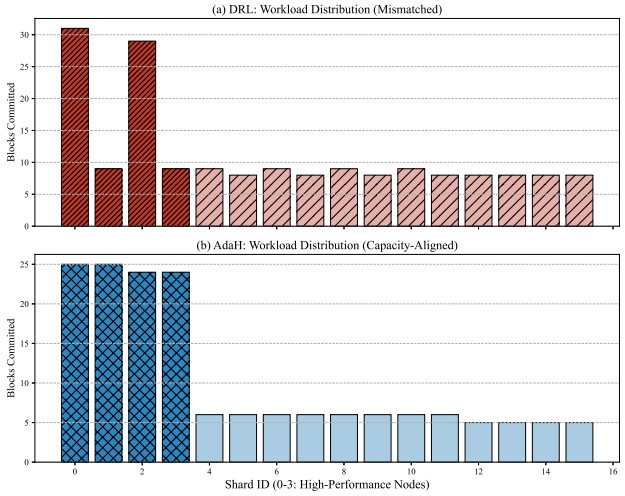


Fig. 6 Workload distribution and block production frequency across physical nodes.

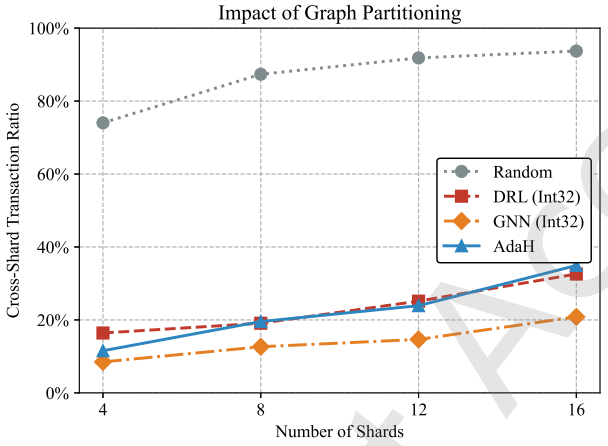


Fig. 7 Cross-Shard Transaction (CST) Ratio. AdaH-Chain trades a moderate communication penalty for strictly matched hardware utilization.

because fast nodes overwhelm slower ones with cross-shard relays. Adding more strong nodes consistently boosts the throughput of AdaH-Chain, while DRL remains bottlenecked by its slowest shard. Under a severe 5,000 ms network delay, the DRL throughput collapses by 25%, whereas AdaH-Chain maintains stability above 160 TPS. The reconfiguration engine exhibits a linear $\mathcal{O}(|A|)$ complexity, effectively processing 1,000,000 accounts asynchronously in 13.4 seconds.

Summary: These results demonstrate the resilience of AdaH-Chain. Confining dense subgraphs within isolated high-performance shards and decoupling routing computation to asynchronous epoch boundaries ensures robust liveness under highly adverse conditions.

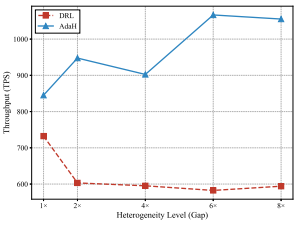


Fig. 8 Throughput sensitivity to varying heterogeneity levels ($1\times$ to $8\times$).

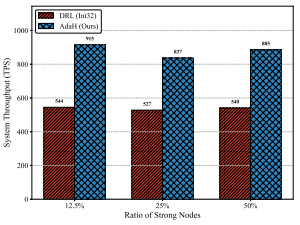


Fig. 9 System throughput under varying proportions of high-performance nodes.

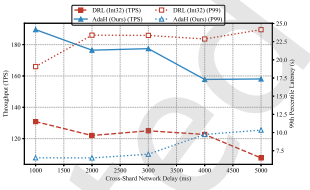


Fig. 10 Robustness under varying network delays (up to 5,000 ms).

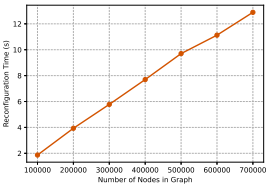


Fig. 11 Scalability of the reconfiguration phase across increasing network sizes.

7 Related Work

Existing literature on blockchain scalability and logical-layer protocols primarily falls into three paradigms: topology-oblivious sharding, AI-driven routing, and heterogeneity-aware consensus mechanisms.

7.1 Topology-Oblivious and DRL-Based Sharding

Early sharding designs prioritize cryptographic security by assuming homogeneous infrastructure. Probabilistic protocols, including OmniLedger [12] and Elastico [22], apply Distributed Randomness Generation to assign accounts to discrete shards. This randomized mapping severs topological dependencies. Breaking temporal locality pushes the cross-shard communication ratio excessively high, forcing the network to rely on expensive locking mechanisms that severely degrade global throughput.

To dynamically adapt to network conditions, recent architectures utilize State-Decision Learning. Protocols such as SPRING [1] and AERO [2] train deep reinforcement learning agents to balance historical transaction locality with real-time shard queue lengths. These learning-based routing methods excel at macro-level workload scheduling. In parallel with computational scheduling, balancing the physical state storage across shards is an equally essential optimization dimension. Frameworks such as Tangram [23] address this structural requirement by enabling dynamic storage extension and balanced state placement to prevent local capacity saturation. However, regarding topological transaction routing, these learning models lack the structural capacity to

extract high-dimensional topological features from the transaction graph. Furthermore, their underlying inference relies on hardware-dependent floating-point libraries, introducing non-determinism during decentralized execution.

7.2 Graph Representation Sharding and The Determinism Gap

To overcome the structural limitations of DRL, advanced protocols introduce Graph Neural Networks to directly optimize the routing topology. Methods utilizing GNNs and graph partitioning algorithms [3, 4] represent historical interactions as weighted graphs to accurately partition densely connected communities.

The primary vulnerability of these continuous models is their reliance on physical hardware-accelerated floating-point arithmetic. Traditional smart contracts maintain absolute execution safety by strictly operating within an integer-based virtual machine sandbox. GNN models must bypass these restrictions to process high-dimensional features. In a strict state machine replication environment, microscopic floating-point deviations generated by disparate instruction set architectures lead to permanent consensus failures.

7.3 Heterogeneity Management in Distributed Systems

Beyond basic consensus, blockchain technology is increasingly utilized to construct reliable evaluation and decentralized reputation systems, effectively preventing malicious data modifications and ensuring accountability in various applications [24, 25]. Building upon this concept within consensus networks, mitigating hardware disparity remains a fundamental challenge. Yu et al. [26] and Huang et al. [27] proposed RepuCoin and ContribChain to integrate reputation scoring to weight consensus participation. They grant higher leader-election probabilities to high-throughput validators. This weighting accelerates network centralization, as powerful nodes accumulate disproportionate block rewards and influence. This mechanism marginalizes weaker participants and leads to excessive power concentration among high-performance nodes.

Alternative models like PoDL [28] attempt to exploit hardware variations by replacing traditional consensus hashing with computational tasks (e.g., deep learning training) routed to GPU accelerators. Similarly, blockchain architectures have been leveraged to securely verify and manage outsourced computation tasks in wireless and edge environments [29]. However, these designs leave the core smart contract evaluation bottleneck unresolved. AdaH-Chain preserves the fundamental one-CPU-one-vote democratic consensus model [30] and differentially routes the actual processing workload. This workload routing mechanism allows the network to extract substantial throughput gains from hardware accelerators without concentrating protocol governance.

Table 3 Comparison of AdaH-Chain with State-of-the-Art Sharding Paradigms. Existing baselines assume homogeneity or rely on unsafe floating-point heuristics. In contrast, AdaH-Chain achieves hardware-aligned efficiency, deep topological extraction, and strict consensus safety.

Framework	Heterogeneity Support	Consensus Determinism	Capacity Awareness	Topology Awareness
OmniLedger (Random)	×	Integer (Safe)	×	×
SPRING / AERO (DRL)	×	Float (Unsafe)	Queue-based	×
Native GNN	×	Float (Unsafe)	×	✓
AdaH-Chain (Ours)	✓	**Int32 (Bit-wise Exact)**	✓($L \propto C$)	✓

7.4 Summary and Design Positioning

Table 3 provides a qualitative comparison between AdaH-Chain and state-of-the-art protocols. Classical graph-based methods optimize for topological locality, and modern DRL approaches offer dynamic adaptability. Both paradigms lack the architectural determinism required for heterogeneous consensus. Existing solutions either treat all nodes equally (triggering the straggler effect) or rely on non-deterministic floating-point arithmetic (compromising safety). AdaH-Chain resolves these structural flaws. It simultaneously enforces Cross-ISA bit-wise consistency and capacity-aligned workload distribution, ensuring robust performance scaling in mixed-hardware environments.

8 Conclusion

This paper resolves the structural conflicts between hardware heterogeneity and consensus determinism in modern sharded blockchains. Conventional smart contracts maintain execution safety by enforcing strict integer logic. Modern sharding protocols increasingly deploy Graph Neural Networks to optimize routing topologies. Forcing these hardware-dependent continuous models onto mixed CPU-GPU clusters inevitably triggers arithmetic state divergence, breaking the fundamental rules of state machine replication. Furthermore, existing protocols suffer from performance-agnostic designs, causing severe straggler bottlenecks when distributing uniform workloads across diverse physical architectures.

To overcome these critical vulnerabilities, we propose the AdaH-Chain framework. We build a Cross-ISA Deterministic Execution Engine utilizing a symmetric fixed-point quantization mechanism. This mathematical transformation effectively eliminates floating-point discrepancies, explicitly isolating the GNN topology extraction from underlying hardware arithmetic variations. Building upon this safe execution

layer, we introduce a Capacity-Aware GNN Sharding algorithm. This mechanism maps dense transaction subgraphs directly to verified physical hardware limits. Evaluations using Ethereum Mainnet traces confirm that our architecture maintains strict consensus safety while preserving 94.7% of the original topological clustering accuracy. The system effectively neutralizes the straggler effect, delivering a peak throughput of 4,800 TPS. This achieves a 4.8× improvement over the randomized sharding baseline and a nearly 2× gain over state-of-the-art learning protocols, ensuring robust performance scaling in highly heterogeneous environments. Future research will explore verifiable stateless off-chain processing mechanisms [31] to further compress cross-shard dependencies and minimize on-chain execution overhead.

Acknowledgment

Yongjiao Sun is supported by the Fundamental Research Funds for the Central Universities (N25ZLH006). Yishu Wang is supported by the National Key R&D Program of China (2024YFE0209000) and the NSFC (62302084, U2441237). Hangxu Ji is supported by the NSFC (62402096) and the Guangdong Basic and Applied Basic Research Foundation (2023A1515110268).

Declaration of competing interest

The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] P. Li, M. Song, M. Xing, Z. Xiao, Q. Ding, S. Guan, and J. Long, SPRING: improving the throughput of sharding blockchain via deep reinforcement learning based state placement, in *WWW*. ACM, 2024, pp. 2836–2846.

[2] M. Song, P. Li, B. Zhou, S. Yin, Z. Xiao, and J. Long, AERO: enhancing sharding blockchain via deep reinforcement learning for account migration, in *WWW*. ACM, 2025, pp. 706–716.

[3] X. Ye, Q. Di, X. Du, A. Bao, and S. Xing, Dual-gru-gnn for dynamic blockchain network sharding, in *2024 9th International Conference on Big Data Analytics (ICBDA)*. IEEE, 2024, pp. 26–33.

[4] H. Li, D. Wang, H. Zhi, Y. Wang, T. Yang, and J. Song, A dynamic sharding scheme for blockchain based on graph partitioning, in *Blockchain*. IEEE, 2024, pp. 286–293.

[5] M. A. Walker, A. Dubey, A. Laszka, and D. C. Schmidt, Platibart: a platform for transactive iot blockchain applications with repeatable testing, in *Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things*, 2017, pp. 17–22.

[6] G. Wood *et al.*, Ethereum: A secure decentralised generalised transaction ledger, *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[7] H. V. Pham, S. Qian, J. Wang, T. Lutellier, J. Rosenthal, L. Tan, Y. Yu, and N. Nagappan, Problems and opportunities in training deep learning software systems: An analysis of variance, in *ASE*. IEEE, 2020, pp. 771–783.

[8] F. B. Schneider, Implementing fault-tolerant services using the state machine approach: A tutorial, *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990.

[9] M. Zamani, M. Movahedi, and M. Raykova, Rapidchain: Scaling blockchain via full sharding, in *CCS*. ACM, 2018, pp. 931–948.

[10] H. Huang, Z. Yin, Q. Chen, J. Zheng, X. Luo, G. Ye, X. Peng, Z. Zheng, and S. Guo, Brokerchain: A blockchain sharding protocol by exploiting broker accounts, *IEEE Trans. Netw.*, vol. 33, no. 4, pp. 1930–1945, 2025.

[11] J. Dean and L. A. Barroso, The tail at scale, *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[12] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, Omniledger: A secure, scale-out, decentralized ledger via sharding, in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2018, pp. 583–598.

[13] F. Erfan and H. Mala, Secure and efficient publicly verifiable outsourcing of matrix multiplication in online mode, *Clust. Comput.*, vol. 23, no. 4, pp. 2835–2845, 2020.

[14] S. Dai, J. Yang, W. Cui, Y. Fang, and Y. Lu, CROSC: compilation-runtime joint optimization for fast smart contract execution, *IEEE Trans. Computers*, vol. 74, no. 9, pp. 2962–2976, 2025.

[15] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 2018.

[16] A. Rossberg, B. L. Titzer, A. Haas, D. L. Schuff, D. Gohman, L. Wagner, A. Zakai, J. F. Bastien, and M. Holman, Bringing the web up to speed with webassembly, *Commun. ACM*, vol. 61, no. 12, pp. 107–115, 2018.

[17] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum, Optimizing space amplification in rocksdb, in *CIDR*. www.cidrdb.org, 2017.

[18] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[19] M. Peruchini and J. M. Teixeira, Exploratory analysis of logical and intuitive reasoning in large language models, *Int. J. Data Sci. Anal.*, vol. 22, no. 1, p. 15, 2026.

[20] H. Huang, G. Ye, Q. Chen, Z. Yin, X. Luo, J. Lin, T. Li, Q. Yang, and Z. Zheng, Blockemulator: An emulator enabling to test blockchain sharding protocols, *CoRR*, vol. abs/2311.03612, 2023.

[21] P. J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[22] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and

P. Saxena, A secure sharding protocol for open blockchains, in *CCS*. ACM, 2016, pp. 17–30.

[23] H. Xu, J. Zhang, X. Liu, Z. Yu, T. Fan, B. Chen, and K. Li, Tangram: Enabling efficient and balanced dynamic storage extension on sharding blockchain systems, *IEEE Trans. Computers*, vol. 74, no. 6, pp. 2031–2044, 2025.

[24] Z. Zhou, M. Wang, C. Yang, Z. Fu, X. Sun, and Q. M. J. Wu, Blockchain-based decentralized reputation system in e-commerce environment, *Future Gener. Comput. Syst.*, vol. 124, pp. 155–167, 2021.

[25] Z. Zhou, M. Wang, Z. Ni, Z. Xia, and B. B. Gupta, Reliable and sustainable product evaluation management system based on blockchain, *IEEE Trans. Engineering Management*, vol. 71, pp. 12 259–12 271, 2024.

[26] J. Yu, D. Kozhaya, J. Decouchant, and P. J. E. Veríssimo, Repucoin: Your reputation is your power, *IEEE Trans. Computers*, vol. 68, no. 8, pp. 1225–1237, 2019.

[27] X. Huang, W. Jie, S. Zhang, H. Yang, W. Qiu, Q. Zhang, H. Huang, Z. Xiong, S. Tang, H. Zheng, and Z. Zheng, Contribchain: A stress-balanced blockchain sharding protocol with node contribution awareness, in *INFOCOM*. IEEE, 2025, pp. 1–10.

[28] C. Chenli, B. Li, Y. Shi, and T. Jung, Energy-recycling blockchain with proof-of-deep-learning, in *IEEE ICBC*. IEEE, 2019, pp. 19–23.

[29] Z. Zhou, Y. Wan, Q. Cui, K. Yu, S. Mumtaz, C. Yang, and M. Guizani, Blockchain-based secure and efficient secret image sharing with outsourcing computation in wireless networks, *IEEE Trans. Wirel. Commun.*, vol. 23, no. 1, pp. 423–435, 2024.

[30] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Self-published, Tech. Rep., 2008, available at: https://bitcoin.org/bitcoin.pdf.

[31] D. Hu, J. Wang, H. Xu, X. Liu, and W. Qu, Rollshard: Atomic multi-shard transactions via verifiable stateless off-chain processing, *IEEE Transactions on Computers*, pp. 1–16, 2026.

Author biography

Ziyao Wang He is currently working toward the Ph.D. degree with the School of Computer Science and Engineering, Northeastern University, Shenyang, China. His current research interests include blockchain technology.

Yishu Wang Yishu Wang received the PhD degree from Northeastern University, China. She is currently a Lecturer with the School of Computer Science and Engineering, Northeastern University. Her research interests include graph data management, graph databases, and spatio-temporal data analysis.