

Efficient Distributed Ledger Techniques with Applications to Distributed Machine Learning

Amirhossein Taherpour

Submitted in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy  
under the Executive Committee  
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2025

© 2025

Amirhossein Taherpour

All Rights Reserved

## **Abstract**

Efficient Distributed Ledger Techniques with Applications to Distributed Machine Learning

Amirhossein Taherpour

The full potential of large-scale decentralized systems, particularly in the Internet of Things and distributed learning, is currently constrained by the persistent trilemma of security, scalability, and interoperability that limits the adoption of distributed ledger technologies. This dissertation confronts these challenges by introducing a portfolio of application-driven architectures and consensus mechanisms, beginning with the fortification of the foundational ledger layer. This includes a high-throughput coded blockchain tailored for the resource constraints of IoT devices and a hybrid architecture that merges sharding with a directed acyclic graph to deliver the fast, accurate consensus required by both real-time data streams and iterative machine learning updates. Building on this foundation, the work presents a scalable interoperability framework for the Web3 vision, using a directed acyclic graph as a common communication substrate for multiple blockchains. The practical applicability of these distributed ledger technologies concepts are then demonstrated through two secure, decentralized federated learning frameworks that directly address the bottlenecks in distributed learning: one integrates a directed acyclic graph sidechain ledger with zero-knowledge proofs for privacy-preserving model validation, and the other synergizes a peer-to-peer gossip protocol with a directed acyclic graph-based control plane using virtual voting to achieve Byzantine fault tolerance. Collectively, this research delivers a portfolio of validated, high-performance frameworks that are not just theoretical constructs, but application-driven solutions designed to

enable the next generation of secure, scalable, and interoperable decentralized systems for IoT and distributed learning.

## Table of Contents

Acknowledgments . . . . .	xiv
Dedication . . . . .	xvi
Introduction or Preface . . . . .	1
Chapter 1: Introduction . . . . .	1
1.1 Core Properties of a Robust Verifiable Substrate . . . . .	3
1.1.1 Security . . . . .	3
1.1.2 Scalability . . . . .	3
1.1.3 Interoperability . . . . .	4
1.2 Thesis Contributions and Roadmap . . . . .	4
1.2.1 Chapter 2: A High-throughput and Secure Coded Blockchain for IoT . . . . .	5
1.2.2 Chapter 3: HybridChain: Fast, Accurate, and Secure Transaction Processing with Distributed Learning . . . . .	5
1.2.3 Chapter 4: SPID-Chain: A Smart Contract-Enabled, Polar-Coded Interoper- able DAG Chain . . . . .	6
1.2.4 Chapter 5: ZK-HybridFL: Zero-Knowledge Proof-Enhanced Hybrid Ledger for Federated Learning . . . . .	6
1.2.5 Chapter 6: Secure Decentralized Federated Learning via Gossip and Virtual Voting . . . . .	7

Chapter 2: A High-throughput and Secure Coded Blockchain for IoT . . . . .	8
2.1 Introduction . . . . .	8
2.2 System Descriptions and Background . . . . .	12
2.2.1 Overview of Coded vs. Uncoded Blockchain Architectures . . . . .	12
2.2.2 Mining Process in IoT Uncoded Blockchains . . . . .	14
2.2.3 Features of LCC-based Coded Blockchains . . . . .	18
2.2.4 Challenges in Uncoded and Coded IoT Blockchains . . . . .	19
2.3 Proposed Coded IoT Blockchain . . . . .	19
2.3.1 Problem Statement . . . . .	20
2.3.2 Background on Rateless Coded Blockchain . . . . .	22
2.3.3 Overview of the Mining Process . . . . .	23
2.3.4 Proposed Mining Procedure . . . . .	25
2.4 Transaction Assignment . . . . .	29
2.4.1 Transaction Selection . . . . .	29
2.4.2 Miner Assignment . . . . .	33
2.5 Comparison with Existing Coded Blockchains . . . . .	35
2.5.1 Simulation Setup . . . . .	35
2.5.2 Storage . . . . .	39
2.5.3 Decentralization . . . . .	41
2.5.4 Throughput . . . . .	43
2.5.5 Security . . . . .	46
2.6 Conclusions . . . . .	51

Chapter 3: HybridChain: Fast, Accurate, and Secure Transaction Processing with Distributed Learning . . . . .	53
3.1 Introduction . . . . .	53
3.2 Backgrounds . . . . .	56
3.2.1 Existing Approaches . . . . .	56
3.2.2 Drawbacks of Existing Schemes and Motivation of Proposed Approach . . . . .	59
3.3 HybridChain . . . . .	61
3.3.1 Overview . . . . .	61
3.3.2 Attribute Vector . . . . .	64
3.3.3 Transaction Validation Procedure . . . . .	65
3.3.4 Updating Rules Inspired by Distributed Learning . . . . .	69
3.4 Security Analysis and Comparison . . . . .	72
3.4.1 Replay Attack . . . . .	72
3.4.2 Message Withholding Attack . . . . .	73
3.4.3 Orphanage Attack . . . . .	74
3.4.4 Routing Attack . . . . .	75
3.5 Simulation Results . . . . .	77
3.5.1 Simulation Setups . . . . .	77
3.5.2 Results . . . . .	81
3.6 Conclusions . . . . .	88
Chapter 4: SPID-Chain: A Smart Contract-Enabled, Polar-Coded Interoperable DAG Chain . . . . .	91
4.1 Introduction . . . . .	91
4.1.1 Related Works . . . . .	92

4.1.2	Motivation and Contributions . . . . .	93
4.2	SPID-Chain . . . . .	96
4.2.1	System Overview . . . . .	96
4.2.2	SPID-Chain Inter-consensus . . . . .	98
4.3	Intra-Consensus in SPID-Chain . . . . .	101
4.3.1	EDSC in SPID-Chain . . . . .	101
4.3.2	Smart Contracts and Consensus on Events . . . . .	103
4.4	SPID-Chain for Cryptocurrency Web3 Networks . . . . .	106
4.4.1	Balance Checking . . . . .	106
4.4.2	Coded Verification . . . . .	109
4.4.3	Structure of the Smart Contracts for Token Transferring . . . . .	112
4.4.4	Stages of SPID-Chain . . . . .	116
4.5	Simulations . . . . .	119
4.5.1	Simulation Setup . . . . .	119
4.5.2	Results . . . . .	121
4.6	Conclusion . . . . .	132
Chapter 5: ZK-HybridFL: Zero-Knowledge Proof-Enhanced Hybrid Ledger for Federated Learning . . . . . 133		
5.1	Introduction . . . . .	133
5.1.1	State-of-the-art and Challenges . . . . .	134
5.1.2	Contributions . . . . .	135
5.2	ZKP-based Consensus Mechanism of ZK-HybridFL . . . . .	136
5.2.1	ZKP . . . . .	136

5.2.2	DAG . . . . .	142
5.2.3	Challenge Mechanism . . . . .	145
5.2.4	Extended ZKP Defenses . . . . .	147
5.3	EDSC Mechanism of ZK-HybridFL . . . . .	150
5.3.1	Smart Contracts in ZK-HybridFL . . . . .	150
5.3.2	Oracle-assisted Sidechain . . . . .	153
5.3.3	Adaptations for Extended ZKP . . . . .	157
5.4	The ZK-HybridFL Procedure and Analysis . . . . .	159
5.4.1	Workflow of ZK-HybridFL . . . . .	159
5.4.2	Comparison with ChainFL . . . . .	163
5.5	Simulation Results . . . . .	167
5.5.1	Simulation Setup . . . . .	168
5.5.2	Learning Tasks . . . . .	169
5.5.3	Results and Analysis . . . . .	171
5.5.4	Additional Simulations . . . . .	179
5.5.5	Simulations of Distributed Ledger Perspectives . . . . .	183
5.5.6	Zero-Knowledge GRU Inference with Recursive Folding . . . . .	185
5.5.7	Extended Security Analysis . . . . .	188
5.6	Conclusions . . . . .	192
Chapter 6: Secure Decentralized Federated Learning via Gossip and Virtual Voting . . . . .		207
6.1	Introduction . . . . .	207
6.1.1	Related Works . . . . .	208

6.1.2	Motivation and Contributions . . . . .	210
6.2	Federated Learning Protocols . . . . .	212
6.2.1	Conventional FL . . . . .	212
6.2.2	Gossip-based Decentralized FL . . . . .	213
6.2.3	FL over Blockchain . . . . .	215
6.3	Gossip-based FL Over Hashgraph DAG (gspDAG-FL) . . . . .	216
6.3.1	System Architecture . . . . .	216
6.3.2	Light-node Duties . . . . .	218
6.3.3	Full-node Duties . . . . .	219
6.3.4	Workflow of DFL over Hashgraph DAG . . . . .	220
6.4	Model Validation and Consensus . . . . .	221
6.4.1	Consensus Process by Full Nodes . . . . .	221
6.4.2	Validations . . . . .	226
6.5	Simulation Results . . . . .	228
6.5.1	Simulation Setup . . . . .	229
6.5.2	Learning Tasks . . . . .	234
6.5.3	Results and Analysis . . . . .	236
6.6	Conclusions . . . . .	242
	References . . . . .	252

## List of Figures

1.1	The conceptual architecture of this dissertation. The research is rooted in the domain of <i>decentralized systems</i> , with a focus on enabling applications in <i>IoT</i> and <i>distributed learning</i> . This is achieved by engineering a robust <i>verifiable substrate (DLT)</i> that balances the core properties of <i>security</i> , <i>scalability</i> , and <i>interoperability</i> . A specialized <i>toolkit</i> of advanced methods is developed and applied to realize the contributions presented in each of the thesis <i>chapters</i> . . . . .	2
2.1	Block coding in the proposed scheme using a systematic raptor code. . . . .	24
2.2	Plot of the reward function $r_j(\mathbf{v}, \mathbf{a}, f_j)$ . . . . .	31
2.3	The modified robust soliton distribution used in the LT code ( $\bar{W} = 1000, \delta = 0.15, c = 0.5$ ). . . . .	37
2.4	Storage usage fraction comparison with $N = 1000, \lambda_l = 4$ , and $\lambda_e = 10$ . . . . .	40
2.5	The comparison of decentralization between LCB and the proposed scheme with $N = 500, \lambda_l = 10, \lambda_e = 20$ and for 500-time epochs, a) Gini coefficient, and b) entropy. 43	43
2.6	Throughput versus the number of miners $N$ with the percentage of dishonest miners $\mu = 30\%$ . . . . .	45
2.7	Throughput versus the percentage of dishonest miners $\mu$ with $N = 500$ miners. . . . .	46
2.8	The effect of discrepancy attack on the throughput based on the number of the miners ( $N$ ) with $\mu = 20\%$ . . . . .	51
2.9	The effect of stragglers on the throughput based on the maximum percentage of the straggler miners with $N = 500$ . . . . .	51

3.1	Transaction Processing in HybridChain at Epoch $n$ . Green circles represent validated transactions, red circles denote rejected ones, and yellow circles indicate transactions under verification. Directed edges show witness relationships, simplifying the DAG structure. Brown ovals represent communities responsible for parallel and sharded transaction processing, highlighting HybridChain’s integrated approach combining sharding and DAG for efficient, secure consensus. . . . .	64
3.2	Accuracy of transaction validity classification under different values of dishonest validators ( $\tau$ ) and number of validators $M$ for HybridChain compared to Random Forest. . . . .	82
3.3	Latency distribution with incoming transactions rate $\gamma = 3 \times 10^4$ , $M = 1000$ validators, and $\tau = 20\%$ . . . . .	82
3.4	Throughput and accuracy based on percentage of dishonest validators, with incoming transactions rate $\gamma = 6000$ and $M = 1000$ validators. . . . .	85
3.5	Throughput and accuracy based on percentage of dishonest validators, with incoming transactions rate $\gamma = 1200$ and $M = 1000$ validators. . . . .	86
3.6	Scalability based on the rate of incoming transactions ( $\gamma$ ), with $\tau = 20\%$ , and $M = 1000$ . . . . .	88
4.1	Evolution of a DAG ledger over two consecutive epochs. Nodes are color-coded to indicate their status: yellow for unconfirmed, green for confirmed, and red for tips. The blue text represents the AW associated with each node. . . . .	101
4.2	Intra-consensus throughput with $N = 10$ , $n = 100$ , and for varying straggler worker percentages $\lambda$ and incoming block rates. . . . .	122
4.3	Intra-consensus scalability with $\lambda = 10\%$ , 12 blocks per minute for rate of incoming blocks, and for varying numbers of nodes within each blockchain and different numbers of blockchains $N$ . . . . .	123
4.4	Inter-consensus throughput with $\lambda = 10\%$ , $N = 10$ , $K = 2$ , $\mu_{\text{crit}} = 50\%$ , and for varying rates of incoming blocks and spamming rates. . . . .	124
4.5	Inter-consensus scalability for varying numbers of blockchains $N$ and worker nodes within each blockchain $n \in \{100, 200\}$ with fixed incoming block rate 10 blocks per min, $\mu = 20\%$ , $\lambda = 10\%$ , $K = 2$ , and $\mu_{\text{crit}} = 50\%$ . . . . .	126
4.6	Network decentralization measured by Gini coefficient for different $K$ and $\mu_{\text{crit}}$ values and spamming rates ( $\mu$ ), with $N = 10$ , $n = 100$ , and $\lambda = 10\%$ . . . . .	128

4.7	Impact of varying spamming rate ( $\mu$ ) on DAG ledger health, demonstrated by tip pool size and inter-chain block finality time. Simulation parameters: $N = 10$ blockchains, $n = 100$ worker nodes with $\lambda = 10\%$ , $\eta = 67\%$ , $K = 2$ , $\mu_{\text{crit}} = 50\%$ and $\mu \in \{35\%, 55\%\}$ , $K = 4$ , $\mu_{\text{crit}} = 75\%$ and $\mu \in \{60\%, 80\%\}$ . . . . .	130
4.8	Performance of SPID-Chain in detecting double-spend blocks with $N = 10$ , $\lambda = 10\%$ , and $\eta = 67\%$ . . . . .	131
5.1	Evolution of a DAG ledger over two consecutive epochs. Nodes are color-coded to indicate their status: red for tips, yellow for unconfirmed, and green for confirmed. The blue text represents the AW associated with each node. . . . .	145
5.2	Full nodes (FNs) maintain a complete DAG (FN_DAG) and sidechain (FN_SC) while handling oracle functions. In contrast, light nodes (LNs) use a trimmed DAG (LN_DAG) and lightweight sidechain (LN_SC), with LN_M serving as LN's core controlling module. . . . .	163
5.3	ZK-HybridFL workflow (1–12): 1 Aggregate IDs; 2 Fetch blocks; 3 Local train; 4 Bundle proofs; 5 Submit bundle; 6 Committee admit; 7 Fetch ZKP tips; 8 Validate ZKPs; 9 Parent selection; 10 Attach block; 11 Update weights; 12 Aggregate & reward. Challenge loop (i–iv): (i) Fetch proof; (ii) Local GRA; (iii) Submit proof; (iv) Reward / slash. . . . .	193
5.4	Training loss curves for Blade-FL, ChainFL, and ZK-HybridFL in a network with $n = 15$ nodes, $\mu = 20\%$ adversaries, $\gamma = 10\%$ lazy nodes, $R = 5$ and $B = 50$ . . . . .	194
5.5	Number of detected invalid models over training epochs corresponding to Fig. 5.4. . . . .	195
5.6	Model performance of ZK-HybridFL, ChainFL, and Blade-FL versus number of nodes $n$ with $\mu = 15\%$ adversarial and $\gamma = 15\%$ lazy nodes. . . . .	196
5.7	Model performance of ZK-HybridFL, ChainFL, and Blade-FL versus adversarial node ratio $\mu$ with $n = 15$ nodes and $\gamma = 15\%$ lazy nodes. . . . .	197
5.8	Model performance of ZK-HybridFL, ChainFL, and Blade-FL versus lazy node ratio $\gamma$ with $n = 15$ nodes and $\mu = 15\%$ adversarial node. . . . .	198
5.9	Stake-Weighted vs Uniform Averaging ( $\gamma = 0.15$ , $\mu = 0.15$ , $n = 15$ ). (a) Task 1 Loss, (b) Task 1 Accuracy, (c) Task 2 Perplexity, (d) Mixed-fault Robustness. . . . .	199
5.10	Extended proof bundle under a mixed-fault setting ( $\gamma = \mu = 0.15$ ). Solid: extended ZK-HybridFL; dashed: ChainFL; dash-dot: Blade-FL. . . . .	200

5.11	Proof-level audit of the mixed-fault run ( $\gamma = \mu = 0.15$ ; five faulty nodes, one update per round).	201
5.12	Lazy-only scenario ( $\gamma = 0$ ). Solid: ZK-HybridFL; dashed: ChainFL; dash-dot: Blade-FL.	202
5.13	Utility-preserving adversaries for ChainFL: high public-set utility yet harmful global impact (mixed-fault setting, $\gamma=0.10$ , $\mu=0.20$ , $n=20$ ).	203
5.14	Latency of Blade-FL, ChainFL, and ZK-HybridFL versus the number of nodes $n$ with $\mu = 20\%$ adversarial nodes and $\gamma = 10\%$ lazy nodes.	204
5.15	Throughput of Blade-FL, ChainFL, and ZK-HybridFL versus the number of nodes $n$ with $\mu = 20\%$ adversarial nodes and $\gamma = 10\%$ lazy nodes.	205
5.16	Scalability of Blade-FL, ChainFL, and ZK-HybridFL versus the number of nodes $n$ with $\mu = 20\%$ adversarial nodes $n$ and $\gamma = 10\%$ lazy nodes.	206
6.1	Communication protocols for different FL approaches.	216
6.2	Illustration of our proposed Hashgraph DAG with $N = 4$ nodes (with nodes connectivity shown as red lines) over five ripples for epoch $t$ . Each circle $e_i(r)$ denotes the event created by node $i$ in ripple $r$ . Cross-column solid edges represent parents (gossips among nodes). The fill color encodes the event's moment value ( $\mathcal{M} = 1$ in blue, $\mathcal{M} = 2$ in green, $\mathcal{M} = 3$ in orange). Red-outlined events are the first event of each moment for each node that are eligible to vote, and dashed circles mark empty events with no payload forwarded.	243
6.3	Training loss versus epoch for gspDAG-FL under four combinations of adversarial ratio $\mu$ and lazy ratio $\gamma$ . (a) Task 1; (b) Task 2.	244
6.4	Number of ripples $R_t$ versus epoch $t$ with $N=15$ , $Q=11$ , and $R^{\max}=13$ . Curves compare clean, adversary-heavy, and lazy-heavy regimes under the two tasks.	245
6.5	Model performance vs. number of nodes under a fixed fault mix with $\mu = 15\%$ adversarial and $\gamma = 10\%$ lazy participants.	246
6.6	Model performance at fixed $N = 15$ and $\gamma = 0.10$ while varying the adversarial ratio $\mu$ .	247
6.7	Model performance at fixed $N = 15$ and $\mu = 0.15$ while varying the lazy share $\gamma$ in 5% steps.	248
6.8	Normalized latency scaling with number of nodes $N$ at fixed $\mu = 0.15$ and $\gamma = 0.10$ .	249

6.9	Normalized throughput scaling with number of nodes $N$ at fixed $\mu = 0.15$ and $\gamma = 0.10$ . . . . .	250
6.10	Normalized scalability vs. number of nodes $N$ at fixed $\mu = 0.15$ and $\gamma = 0.10$ . . . . .	251

## List of Tables

2.1	Comparison Between Coded and Uncoded Blockchains . . . . .	15
2.2	Matrix Representation of a Block with Grouped Columns for $L$ (Metadata), $S$ (State Information), and $R$ (Functions) . . . . .	16
2.3	Resistance to various attacks by different coded blockchain schemes. . . . .	49
3.1	Comparison of IOTA, Omniledger, and the proposed HybridChain based on the trilemma of scalability, decentralization, and security. . . . .	61
3.2	Elements of the attribute vector $\mathbf{a}_\ell$ . . . . .	65
3.3	Resistance to various attacks by different distributed ledger schemes. . . . .	76
3.4	Distribution of attribute vector elements for generating valid and invalid transactions. For $a_\ell[3]$ , the generated number is rounded down to find a discrete number of rounds. . . . .	80
4.1	Challenges and SPID-Chain solutions for blockchain interoperability. . . . .	95
4.2	Consensus mechanisms in SPID-Chain. . . . .	97
4.3	Summary of Smart Contracts for Token Transferring . . . . .	116
5.1	Summary of EDSCs for ZK-HybridFL. . . . .	153
5.2	Oracle Committee micro-benchmarks (per approved event). . . . .	156
5.3	Comparison of ChainFL and ZK-HybridFL. . . . .	167
5.4	Proof-generation cost per node (batch size $B = 10$ ). . . . .	177
5.5	Proof verification overhead. . . . .	177

5.6	Incremental proofs: generation and verification cost (same hardware as Tables 5.4–5.5).	178
5.7	Extended-ZKP attack variants.	180
6.1	Detection Efficacy Breakdown for Task 1 (Image Classification)	237
6.2	Detection Efficacy Breakdown for Task 2 (Language Modeling)	237
6.3	Effect of readiness threshold $Q$ and ready fraction $\beta$ on our gspDAG-FL at $N=15$ under three regimes. Each row reports median ripples per epoch $R_t$ and the fraction of epochs that hit the adaptive ripple cap ( $cap\%$ ) for that $(Q, \beta)$ , together with the model performance at convergence for Task 1 and Task 2. For fairness, the ripple cap is set per configuration as $R_t^{\max} = Q+2$ .	238

## **Acknowledgements**

First and foremost, all thanks and praise belong to God for enabling me to continue when I thought I could not, and for illuminating the way with unexpected blessings.

I would like to express my sincere appreciation to my advisor, Prof. Xiaodong Wang, for his patience and for encouraging me to clarify my thinking and strengthen the way I communicate complex ideas. His guidance has shaped how I approach research and how I express it to others. I am grateful for his support and for the opportunity to grow under his supervision throughout my PhD.

I would like to thank the members of my thesis committee: Javad Ghaderi, Micah I. Goldblum, Xiaofan Jiang, and Chong Li. I am grateful for their time and attention, and for their willingness to serve on my committee throughout this process.

I am especially grateful to Prof. John Wright, Vice Chair of the Department of Electrical Engineering, for his support during a critical point in my final year. When I faced unexpected challenges that threatened my ability to continue, he stepped in to help ensure I could complete my degree. I sincerely thank him for that.

Although this thesis marks the formal end of my PhD, I see it as the culmination of something much larger. It is the result of years of life, growth, and support that began long before this program. The following words are not only acknowledgments in the academic sense but also a recognition of the people whose presence, actions, and love made this path possible.

I want to thank my father for the steady ways he showed up for me throughout my childhood and youth. His quiet generosity, from small treats to time spent together, created many of my

happiest early memories, and I am grateful for the kindness and care he gave me. I am also grateful to my sister Maryam, my brother Abolfazl, and my sister Akram for their support during different seasons of my life. Each of them, in their own way, offered encouragement, practical help, and emotional reassurance at times when I needed it. Their presence and care have been part of the foundation that made this journey possible.

I want to thank my mother, who cared for me with all her heart and ability. She expressed her love through countless quiet acts of care. I remember her gentle patience in the early mornings as she helped me get ready, and the way her hands packed my school bag with snacks as carefully as if she were sending a part of herself with me. I remember the school snacks and lunches she prepared for years, especially her halva-lavash loghmehs. The constant care she put into these everyday tasks has stayed with me. I carry this degree in part for her. Through every season of this journey, her steady emotional presence was a pillar I leaned on. Whatever resilience I have is built on that foundation.

I want to thank my brother Abbas for his guidance and steady support throughout my adult life. He has often been the person I turned to for perspective and practical advice, including during the application process that led me here. His encouragement and clear-sighted counsel were important in helping me navigate not only this PhD, but also the broader path that led to it. I could not have earned this PhD without him.

And last but not least, I want to thank my sister Zohreh for her unwavering support during these years. She has been the light when my doubts and disappointments cast everything into darkness. There were times when I felt completely defeated, and the only reason I could keep going was that she reminded me, in her way, that it would be okay. Her presence in my life returned hope to me when I had lost it and gave me the steadiness I needed to continue. I am deeply grateful for the care she has shown and for the belief in me that she never let go of, even when I could not hold it myself.

## **Dedication**

To my mother, Abbas, and Zohreh,  
whose presence made all the difference.

## Chapter 1: Introduction

The last decade has witnessed a surge of interest in decentralized systems, which can be defined as collections of autonomous, interconnected nodes that coordinate to achieve a collective goal without a central authority. Such systems are foundational to modern computing paradigms, enabling everything from large-scale sensor networks in the Internet of Things (IoT) to collaborative model training in distributed learning. In these trustless environments, where no single entity can be relied upon to enforce rules or validate state, the fundamental challenge is to establish a shared, verifiable truth. To achieve this, decentralized systems require a *verifiable substrate*: a foundational layer that allows all participants to agree on the state of the system and the history of its operations in a secure and tamper-evident manner. Distributed Ledger Technologies (DLTs), spearheaded by blockchain, have emerged as the predominant implementation of such a substrate [1].

However, designing a verifiable substrate that is both robust and performant for demanding applications is a non-trivial undertaking. As illustrated in the conceptual hierarchy of Figure 1.1, a successful DLT must be built upon three core properties, or pillars: **Security**, ensuring the integrity of the ledger against malicious actors; **Scalability**, allowing the system to process a high volume of transactions with low latency; and **Interoperability**, enabling seamless and secure communication between disparate ledger systems to form a cohesive ecosystem. This dissertation confronts the challenges inherent in each of these pillars by designing and validating a portfolio of novel DLT frameworks. The contributions herein are achieved through the strategic application of a diverse toolkit of advanced techniques, developed with the specific demands of applications like IoT and distributed learning in mind. The intricate relationships between these application domains, the core DLT principles, the tools employed, and the resulting thesis chapters are holistically mapped in Figure 1.1.

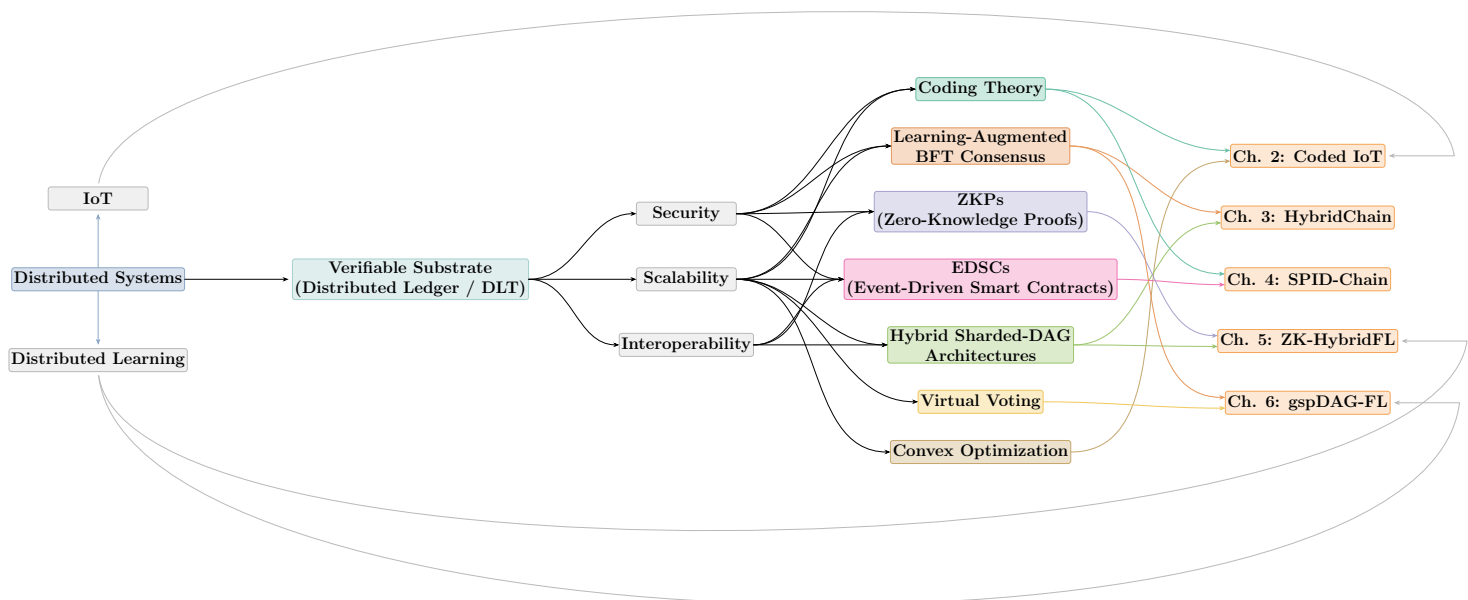


Figure 1.1: The conceptual architecture of this dissertation. The research is rooted in the domain of *decentralized systems*, with a focus on enabling applications in *IoT* and *distributed learning*. This is achieved by engineering a robust *verifiable substrate (DLT)* that balances the core properties of *security*, *scalability*, and *interoperability*. A specialized *toolkit* of advanced methods is developed and applied to realize the contributions presented in each of the thesis *chapters*.

## 1.1 Core Properties of a Robust Verifiable Substrate

This section details the challenges associated with each of the three core DLT properties, reviews the contemporary state of the art in addressing them, and introduces the specific tools leveraged in this thesis to advance these frontiers, as mapped in Figure 1.1.

### 1.1.1 Security

Security is the bedrock of any verifiable substrate. Challenges range from protocol-level Byzantine attacks to application-layer exploits and privacy breaches. A significant portion of modern DLT research is focused on developing robust consensus mechanisms and integrating advanced cryptographic primitives. The state of the art in secure consensus has moved towards Directed Acyclic Graph (DAG) architectures, with protocols like Fides leveraging Trusted Execution Environments (TEEs) [2] and Shoal++ aiming for low-latency algorithmic Byzantine Fault Tolerance (BFT) [3]. In the application space, privacy in Federated Learning (FL) is a major concern due to threats like gradient inversion attacks.

This thesis enhances security through several key tools. We leverage *Coding Theory* to provide resilience and fault tolerance (Chapters 2 and 4). We develop novel *Learning-Augmented BFT Consensus* mechanisms that use distributed learning principles for secure validation (Chapter 3). Crucially, we integrate *Zero-Knowledge Proofs (ZKPs)*, a transformative cryptographic tool that enables the public verification of private computations, thereby eliminating the need for vulnerable public datasets in FL (Chapter 5). Finally, *Event-Driven Smart Contracts (EDSCs)* create a more secure and automated framework for coordinating complex interactions (Chapters 4 and 5).

### 1.1.2 Scalability

Scalability refers to a DLT's ability to process a high volume of transactions with low latency, a primary obstacle for applications in IoT and global-scale systems [4]. Sharding, seen in systems like Omniledger [5], partitions a network for parallel processing but can introduce cross-shard

communication latency, a problem recent protocols like Kronos [6] and DynaShard [7] aim to solve. DAGs offer another path to scalability, but present their own consensus challenges. The broader trend is toward modularity, where blockchain functions are unbundled into specialized layers [8].

This dissertation addresses scalability through a multi-faceted approach. We introduce novel *Hybrid Sharded-DAG Architectures* that synergistically combine the strengths of both paradigms (Chapters 3, 4, and 5). We employ *Virtual Voting*, a gossip-native consensus mechanism inspired by Hashgraph [9], to achieve finality with high speed (Chapter 6). Additionally, techniques like *Coding Theory* and *Convex Optimization* are applied to enhance storage efficiency and optimize resource allocation, further contributing to overall system scalability (Chapter 2).

### 1.1.3 Interoperability

Interoperability, the ability of disparate DLTs to communicate securely, is the cornerstone of the Web3 vision. The bridges connecting these networks, however, have become a primary attack surface, highlighting the fragility of existing solutions [10], [11]. Current approaches range from trusted notary schemes to more decentralized sidechains and relay protocols, each with its own trade-offs.

This thesis contributes a novel framework for robust and scalable interoperability, primarily through *Hybrid Sharded-DAG Architectures* and *EDSCs*. By using a DAG as a common communication substrate for multiple independent blockchains, our architecture facilitates asynchronous and efficient cross-chain interactions. This approach is central to the SPID-Chain framework (Chapter 4), which is designed explicitly to address the challenges of building a secure and functional multi-chain ecosystem.

## 1.2 Thesis Contributions and Roadmap

The preceding sections have outlined the core challenges in DLT design and the advanced tools used in this dissertation to address them. The research is structured as a series of self-contained contributions, each targeting specific problems within the broader landscape, with a focus on

enabling applications in IoT and distributed learning. The remainder of this document details these contributions in full.

### 1.2.1 Chapter 2: A High-throughput and Secure Coded Blockchain for IoT

This chapter proposes a new coded blockchain scheme suitable for the IoT network. In contrast to existing works for coded blockchains, the proposed scheme is more realistic, practical, and secure while achieving high throughput. This is accomplished by: 1) modeling the variety of transactions using a reward model, based on which an optimization problem is solved to select transactions that are more accessible and cheaper computational-wise to be processed together; 2) a transaction-based and lightweight consensus algorithm that emphasizes using the minimum possible number of miners for processing the transactions; and 3) employing Raptor codes with linear-time encoding and decoding, which results in requiring lower storage to maintain the blockchain and having a higher throughput. We provide detailed analysis and simulation results on the proposed scheme and compare it with state-of-the-art coded IoT blockchain schemes, including Polyshard [12] and LCB [13], to show the advantages of our proposed scheme in terms of security, storage, decentralization, and throughput.

### 1.2.2 Chapter 3: HybridChain: Fast, Accurate, and Secure Transaction Processing with Distributed Learning

To fully unlock the transformative power of distributed ledgers, it is crucial to develop innovative consensus algorithms that can overcome the obstacles of security, scalability, and interoperability. This chapter introduces HybridChain, which combines the advantages of sharded blockchain and DAG distributed ledger, and a consensus algorithm that leverages decentralized learning. Our approach involves validators exchanging perceptions as votes to assess potential conflicts between transactions and the witness set. These perceptions collectively contribute to an intermediate belief regarding the validity of transactions. By integrating their beliefs with those of other validators, localized decisions are made, and a final consensus is achieved through a majority vote. Our proposed approach is compared to the existing DAG-based scheme IOTA [14] and the sharded

blockchain Omniledger [5] through extensive simulations. The results show that while IOTA has high throughput at the cost of accuracy and Omniledger achieves stability at the cost of latency, the proposed HybridChain exhibits fast, accurate, and secure transaction processing with excellent scalability.

### 1.2.3 Chapter 4: SPID-Chain: A Smart Contract-Enabled, Polar-Coded Interoperable DAG Chain

As the digital landscape evolves, Web3 has gained prominence, highlighting the critical role of decentralized and interconnected digital ecosystems. This chapter introduces SPID-Chain, a novel interoperability consensus designed for Web3, which employs a DAG of blockchains to facilitate seamless integration across multiple blockchains. Within SPID-Chain, each blockchain maintains its own consensus and processes transactions via an intra-consensus mechanism that incorporates EDSCs and Polar codes for optimized computation distribution. This mechanism is complemented by a division of committee and worker nodes. For inter-blockchain consensus, SPID-Chain utilizes a DAG structure where blockchains append blocks containing cross-chain transactions. Extensive simulations validate the efficacy of our scheme in terms of throughput, scalability, decentralization, and security, showcasing SPID-Chain’s potential to enable fluid interactions and transactions across diverse blockchain networks.

### 1.2.4 Chapter 5: ZK-HybridFL: Zero-Knowledge Proof-Enhanced Hybrid Ledger for Federated Learning

FL enables collaborative model training while preserving data privacy, but existing decentralized approaches face significant challenges in scalability, security, and update validation. This chapter proposes ZK-HybridFL, a secure decentralized FL framework that integrates a DAG ledger with dedicated sidechains and ZKPs for privacy-preserving model validation. By leveraging EDSCs and an oracle-assisted sidechain, our framework efficiently verifies local model updates without revealing sensitive data, while a built-in challenge mechanism detects and mitigates adversarial behavior. Simulation results on image classification and language modeling tasks demonstrate that

ZK-HybridFL achieves faster convergence, higher accuracy, lower latency, and improved resilience against both adversarial and lazy nodes compared with state-of-the-art schemes like Blade-FL [15] and ChainFL [16]. These findings highlight ZK-HybridFL’s potential as a scalable and secure solution for decentralized FL.

### 1.2.5 Chapter 6: Secure Decentralized Federated Learning via Gossip and Virtual Voting

As FL scales to vast networks of devices, the central server in conventional architectures becomes a significant performance bottleneck. Therefore, serverless, gossip-based decentralized FL (DFL) has emerged as a critical paradigm for its superior scalability and efficiency. However, the efficiency of pure gossip protocols comes at the cost of lacking a mechanism for auditable, global consensus on accepted model updates. While ledger-based solutions have been proposed to bring finality to FL, they are often designed for centralized topologies and impose confirmation latencies and broadcast overheads that are fundamentally at odds with the lightweight, peer-to-peer nature of DFL. To resolve this dilemma, we introduce *gspDAG-FL*, a novel framework that synergizes a gossip-based data plane with a DAG-based control plane for consensus. In our architecture, nodes exchange model updates via efficient, one-shot peer-to-peer gossip. Concurrently, a subset of full nodes reconstructs a global *Topology DAG* from lightweight cryptographic proofs of these exchanges. This structure enables asynchronous BFT finality through a Hashgraph-style virtual voting protocol, which finalizes updates without explicit vote messages or linear blocks. Resilience is achieved through a multi-stage defense pipeline: nodes apply magnitude and directional filters to updates during gossip, and after consensus, perform a semantic audit on private validation sets to identify and reject stealthy backdoors before aggregation. We empirically evaluate *gspDAG-FL* on image classification and language modeling, showing it maintains model accuracy and perplexity on par with state-of-the-art ledger-based FL under mixed adversarial and lazy participation. Meanwhile, *gspDAG-FL* delivers lower latency, higher throughput, and better scaling with network size. These results demonstrate that a gossip-native, DAG-assisted control plane can provide robust, auditable consensus in DFL without sacrificing the lightweight, peer-to-peer efficiency of pure gossip protocols.

## **Chapter 2: A High-throughput and Secure Coded Blockchain for IoT**

### **2.1 Introduction**

The Internet of Things (IoT) has become one of the most important technologies in the 21st century, which conceptualizes a hyper connected world where physical things (ranging from small embedded sensors and wearable gadgets to connected vehicles and automated systems) can share and collect data with minimal human intervention. IoT facilitates accomplishing various tasks in people's daily lives and enhances their interactions with their surroundings. However, the increases in size, complexity, and the number of connected devices in IoT networks pose significant challenges in a number of aspects including centralization, authenticity, reliability, anonymity, and security [17]–[19].

Blockchain technology which is a distributed immutable ledger with high transparency and reliability has received considerable attention as one of the promising solutions to some of the above challenges with IoT networks [20]–[22]. Although the blockchain has a number of advantages, the main issue is that finding a proper consensus algorithm for IoT blockchains is challenging since existing consensus algorithms for conventional blockchains, e.g., Proof of Work (PoW), Proof of Stake (PoS), and some others that were proposed later, e.g., practical Byzantine fault tolerance (pBFT), Proof of Authority (PoA) do not meet the stringent requirements of IoT blockchains, i.e., real-time transaction settlement and high security [23]–[28]. Since an IoT blockchain consists of many heterogeneous devices with diverse functionalities, the network could produce millions of transactions in a short period of time. At the same time, some of them have to be processed quickly with the highest accuracy possible. The above-mentioned existing consensus algorithms either do not have the desired high throughput or can achieve the throughput only by sacrificing the network security. Another problem with the existing blockchains is that all data are replicated in each node

and therefore storage becomes a major problem after some time. This issue is more severe with IoT blockchains with millions of transactions and many resource-restricted devices and some of which might be miners as well [29].

Recently coded blockchain techniques have been proposed [12], [20], [29]–[32] to achieve better throughput and security along with reduced size of the blockchain. Some of these works were inspired by coded distributed computing (CDC) [33]–[38]. The basic idea in CDC is that when a computation task needs to be done, working on encoded data provides the opportunity to divide the whole computation into sub-tasks, each of which could be done independently and a subset of the computation results of those sub-tasks suffice to obtain the final result of the original computational task.

In particular, Polyshard [12] makes use of the notion of sharding in blockchain [5], [39]–[42] and employs the Lagrange Coded Computing scheme (LCC) [13]. A sharded blockchain splits a blockchain network into smaller partitions, known as “shards.” Each keeps its own data, which is distinctive and independent from data in other shards; and the nodes in each shard are only responsible for managing transactions in their shard. In Polyshard, to verify blocks generated within a shard, nodes individually compute a polynomial verification function over a coded chain to find an intermediate result, and having intermediate results from all other miners in the network, nodes can reach a consensus about the validity of each block. However, Polyshard suffers from many problems such as, delay, security, and oversimplification of the model [20], [42]–[45]. [44] tries to make the Polyshard more practical by considering a cross-sharded blockchain where nodes that are not in the same shard still can place a transaction in the network, a crucial feature that Polyshard lacks. The Lagrange Coded Blockchain (LCB) [20] is a non-sharded approach that tries to improve Polyshard by using the LCC in both block generation and verification stages. Also, to solve the security issue of Polyshard that arises when a group of miners, called stragglers, are not capable of sending the computation results in time, LCB assigns more than one computation task to miners that are faster and capable of doing more computations. However, the LCB scheme is not storage efficient, which is a significant drawback for IoT networks. Furthermore, it gives rise to some other security issues and

leads to a centralized network which will be discussed in this chapter. While [30] provides a secure naming and storage system using blockchain technology, it is primarily focused on static data storage and naming services, making it unsuitable for high-frequency, real-time transaction processing typical in IoT networks. Moreover, Blockstack's design in [30] does not inherently address the scalability and storage efficiency challenges posed by IoT devices with limited resources, nor does it incorporate advanced coding or sharding techniques to optimize throughput and minimize latency in dynamic IoT environments. In [29], [31], besides proposing a storage efficient blockchain, authors aim to minimize the bootstrapping cost using some coding techniques, while throughput-wise, these schemes do not scale well since these works employ the conventional consensus algorithms like PoW.

From the above discussions, each of the existing works in coded blockchain has some drawbacks. Particularly, when contemplating the formulation of an IoT blockchain framework, certain crucial aspects necessitate meticulous deliberation, thereby unveiling the conspicuous inadequacies prevalent in the aforementioned studies:

1. **Enhanced Efficiency:** The existing blockchains are inefficient in handling the high volume of transactions generated in IoT networks. By designing a new IoT blockchain, we can significantly increase throughput and overcome the limitations of existing systems, ensuring seamless operation and efficient processing of IoT transactions.
2. **Priority-based Transaction Handling:** IoT applications require different levels of transaction priority based on their versatility and importance. A new IoT blockchain can incorporate sophisticated transaction selection and consensus procedures to ensure that critical transactions receive the appropriate priority, enhancing the overall functionality and effectiveness of IoT applications.
3. **Overcoming Scalability, Resource Constraints, and Latency:** Scaling a blockchain to handle the high transaction throughput and storage requirements of IoT data, accommodating resource-constrained devices, and providing low-latency communication are significant challenges.

A new IoT blockchain can address these challenges through innovative solutions, enabling robust scalability, efficient operation on resource-constrained devices, and quick transaction confirmation for real-time IoT applications.

In this chapter, we propose a new coded blockchain for IoT networks that does not suffer from any of the drawbacks mentioned in existing works and considers the above aspects. Here we list the main contributions of this article:

1. We introduce the concept of transaction selection in the context of an IoT network to prioritize different types of transactions. By formulating the priority of transactions and addressing it as a stochastic or deterministic optimization problem, we can identify and select transactions of high priority for processing achieving high throughput in the system.
2. Considering a dynamic IoT network in which nodes may join and leave the network, suitable for IoT scenario, we propose a lightweight consensus algorithm that along with the transaction prioritization and the low-complexity decoding procedure for fetching the required data for verification, help to reduce the futile work and result in a high throughput.
3. We discuss in-depth the notions of security, storage reduction, decentralization, and throughput in coded blockchains and compare our proposed scheme with Polyshard and LCB in terms of these metrics.

The remainder of the chapter is organized as follows. Section 2.2 introduces the blockchain of things system model and the existing coded blockchains. Section 2.3 is an overview of our proposed scheme. In Section 2.4, we provide details on the transaction selection and assignment stage of our proposed scheme. In Section 2.5, through simulations we compare our scheme with Polyshard and LCB schemes in terms of storage reduction, decentralization, throughput, and discuss the security issues. Finally, Section 2.6 concludes the chapter.

## 2.2 System Descriptions and Background

A typical IoT system consists of three elements: 1) IoT devices that produce data. These devices can be in a wide variety [46], including industrial sensors working with Low Power Wide Area Networks (LPWANs), smart retail that uses Bluetooth Low-Energy (BLE), logistics and asset tracking operating with Radio Frequency Identification (RFID), wearables, and smart homes deploying cellular/Wi-Fi connectivity. 2) Base station (BS), which receives the data from IoT devices and aggregates them to form batches that are sent to miners. 3) Miners that verify the validity of given data from the BS, and after reaching a consensus, the validated data are appended to the blockchain in the form of blocks. Next, we explain the mining process in an IoT blockchain in detail.

### 2.2.1 Overview of Coded vs. Uncoded Blockchain Architectures

Blockchains can be fundamentally classified into *uncoded* and *coded* blockchains [47]. This classification is pivotal for understanding the subsequent sections introduced in this chapter.

#### Uncoded Blockchain

In an *uncoded* blockchain architecture, each data block is fully replicated across all network nodes, impacting both storage and retrieval processes.

**Storage.** In a traditional uncoded blockchain, each data block  $B(m)$  is fully replicated and stored by every node within the network. Specifically, for a blockchain consisting of  $t$  blocks, each node maintains the complete set  $\mathbf{B}^i = \{B(1), B(2), \dots, B(t)\}$ . This full replication ensures that every node holds an independent copy of all blocks on the chain. While this approach guarantees data availability, it incurs substantial storage overhead as the blockchain scales. Prominent blockchains, such as Bitcoin and Ethereum, exemplify this limitation, requiring nodes to store hundreds of gigabytes of data, a demand that escalates as new blocks are added, posing significant scalability challenges.

**Recovery and Verification.** Recovery and verification are straightforward in an uncoded blockchain. When a specific block  $B(i)$  is required, each node accesses it directly from its local storage. This process allows for efficient retrieval speed and enables verification without relying on other nodes. For example, to validate a transaction within block  $B(i)$ , a node retrieves  $B(i)$  directly from its local copy  $\mathbf{B}^t$ , eliminating the need for external assistance. However, this efficiency in retrieval comes at the cost of increased storage and communication overhead, as every node must replicate the entire blockchain, leading to significant data redundancy.

### Coded Blockchain

In a *coded* blockchain architecture, each block is divided into smaller fragments and encoded using error correction techniques, distributing these coded fragments across network nodes and providing enhanced storage efficiency and resilience.[48]

**Storage.** In a coded blockchain, each block  $B(m)$  is divided into  $k$  original fragments, denoted as  $\{B(m)_1, B(m)_2, \dots, B(m)_k\}$ . These original fragments are then encoded to produce  $N$  coded fragments, denoted  $\tilde{B}_1(m), \tilde{B}_2(m), \dots, \tilde{B}_N(m)$ , which are distributed across the network nodes. Each node in the network stores only *one coded fragment* of each block, so a node  $j$  would store  $\tilde{B}_j(m)$  for block  $B(m)$  in epoch  $m$ .

These  $N$  coded fragments are constructed as *linear combinations* of the  $k$  original fragments. Specifically, each coded fragment  $\tilde{B}_j(m)$  for node  $j$  is formed by linearly combining the original fragments  $\{B(m)_1, B(m)_2, \dots, B(m)_k\}$  with predetermined coding coefficients. Therefore, node  $j = 1, \dots, N$  stores  $\tilde{B}_j(m)$ , where each  $\tilde{B}_j(m)$  is a coded fragment that can collectively reconstruct  $B(m)$  when any subset of  $k$  fragments is available.

This approach significantly reduces the storage burden on each node, as each node only needs to retain a fraction  $\frac{1}{N}$  of the blockchain data, improving scalability and efficiency over traditional uncoded blockchain systems.

**Recovery and Verification.** In a coded blockchain, recovery of a block  $B(m)$  relies on the distributed coded fragments. To reconstruct the original block for purposes such as transaction

validation or consensus, any node can collect a subset of  $k$  coded fragments from the network's  $N$  nodes. Using a decoding algorithm, such as Reed-Solomon decoding, the node combines these  $k$  coded fragments to reconstruct  $B(m)$  fully. The linear combination encoding provides robustness, as any  $k$  out of  $N$  coded fragments are sufficient for complete recovery, ensuring data availability even if some nodes fail or their data becomes corrupted.

The security parameters of the consensus protocol and the encoding scheme parameters ( $k, N$ ) are conceptually distinct yet indirectly interconnected through system redundancy and data recovery capabilities. The consensus mechanism ensures correct decision-making even in the presence of faulty or adversarial nodes, while the encoding scheme optimizes storage efficiency and data availability via its chosen  $k$  and  $N$  values. Although the consensus fault tolerance does not directly dictate encoding choices, selecting a higher  $N/k$  ratio enhances data redundancy, which in turn bolsters fault tolerance by safeguarding against data loss or corruption amid node failures or malicious activity. This enhanced redundancy ensures reliable reconstruction of original data, thus supporting the accuracy and integrity of consensus outcomes. However, these benefits come with increased storage, communication, and computational overhead, necessitating a balanced, coordinated tuning of  $k$  and  $N$ . Such tuning aligns encoding parameters with security requirements, achieving the desired level of resilience without compromising system efficiency.

Table 2.1 provides a comprehensive comparison between coded and uncoded blockchains, highlighting their distinct characteristics in terms of storage mechanisms, scalability, fault tolerance, data retrieval, redundancy, communication overhead, and implementation complexity[47]. This comparison underscores the coded blockchain's suitability for environments with constrained storage and bandwidth, such as IoT networks, by addressing the storage and communication inefficiencies that limit the scalability of conventional blockchain systems.

### 2.2.2 Mining Process in IoT Uncoded Blockchains

We divide the blockchain time-wise into epochs and assume that one block is appended to the blockchain after each epoch. Suppose, at the beginning of epoch  $t$ , the system consists of

Table 2.1: Comparison Between Coded and Uncoded Blockchains

Aspect	Uncoded Blockchain	Coded Blockchain
Storage Mechanism	Full replication of all blocks by each node.	Storage of encoded fragments; only a subset is required for recovery.
Scalability	Limited by linear growth in storage and computational needs.	Enhanced scalability through distributed storage and processing.
Fault Tolerance	Vulnerable to node failures that affect data availability.	Resilient to node failures, tolerating up to $n - k$ faults.
Data Retrieval	Direct access to complete blocks by each node.	Requires decoding from any $k$ fragments to reconstruct a block.
Redundancy	High redundancy; each node stores the entire blockchain.	Lower redundancy; each node holds only a fraction of the blockchain.
Communication Overhead	Higher bandwidth requirements due to full block transmission.	Reduced bandwidth usage with coded fragment transmission.
Implementation Complexity	Lower complexity due to straightforward data storage.	Higher complexity due to encoding and decoding mechanisms.

$N(t)$  miners. One epoch of the mining process starts with collecting the data from all devices connected to the BS. Denote the matrix of data collected by the BS by  $T(t) = [L(t) \ S(t) \ R(t)]$  that consists of  $n$  transactions, each corresponding to one row. Here,  $L(t)$  encompasses metadata such as transaction ID, sender, recipient, amount, and contract type;  $S(t)$  represents state information detailing contract conditions and post-execution updates; and  $R(t)$  specifies the functions invoked for contract execution.

Matrix-based representations of transactions and blocks are grounded in extensive research within the coded blockchain domain, demonstrating significant improvements in scalability, storage efficiency, and computational performance [29], [49]–[51]. Building upon these established methodologies, our approach tailors matrix structures specifically to address the constraints and requirements of IoT environments.

In our framework, transactions are organized as rows within a transaction matrix  $T(t) = [L(t) \ S(t) \ R(t)]$ . Similarly, blocks are structured as columns within a block matrix  $B(m) = [L^*(m) \ S^*(m) \ R^*(m)]$ , aggregating validated transactions and states for each epoch  $m$ . This organization facilitates efficient encoding and storage across network nodes.

To illustrate, consider Ethereum blockchain where the metadata  $L(t)$  includes transaction hash,

sender and recipient addresses, value transferred, transaction fees, and contract type. The state information  $S(t)$  captures the contract’s current state, such as balances, contract-specific variables, and execution conditions, while  $R(t)$  details smart contract functions invoked, such as `transfer`, `approve`, or `notify`. This alignment ensures compatibility with existing blockchain frameworks, leveraging well-understood structures to enhance our coded blockchain’s functionality.

Table 2.2: Matrix Representation of a Block with Grouped Columns for  $L$  (Metadata),  $S$  (State Information), and  $R$  (Functions)

Transaction	$L$ : Metadata				$S$ : State Information			$R$ : Functions			
	Transaction ID	Sender	Recipient	Amount	Contract	Status	Block Number	Function	Gas	Event	Callback
Transaction 1	0xabc	0xS1	0xR1	2	0xC1	Confirmed	1250	transfer	21000	Transfer	UpdateBalance
Transaction 2	0xdef	0xS2	0xR2	3	0xC2	Pending	1251	approve	30000	Approval	InitiatePayment
Transaction 3	0xghi	0xS3	0xR3	1	0xC3	Executed	1252	notify	25000	Notify	AlertDoctor

Table 2.2 presents a concrete matrix representation of transactions, where each row corresponds to a transaction segmented into Metadata ( $L$ ), State Information ( $S$ ), and Functions ( $R$ ). Specifically, the metadata includes Transaction ID, Sender, Recipient, Amount, and Contract type; the state information captures Contract address, Status (e.g., Pending, Confirmed, Executed), and Block Number; and the functions specify the invoked function (e.g., `transfer`, `approve`, `notify`), Gas allocated, Event triggered, and Callback functions. This structured representation aligns with Ethereum’s blockchain, facilitating efficient encoding, storage, and verification processes essential for our coded blockchain framework.

This matrix-based representation enables encoding over finite fields  $\mathbb{F}_q$ , allowing the application of error-correcting codes that enhance data resilience against loss and attacks, critical for decentralized coded IoT networks. Matrix operations also support parallel processing of transactions, which accelerates verification and consensus through optimized linear algebra techniques, increasing system throughput and benefiting IoT networks with limited computational resources. Encoding blocks as matrices enhances resilience against tampering, ensuring data integrity and supporting robust security protocols. This systematic coding approach further obscures transaction details, adding an additional layer of security

We next illustrate these three components of the transaction data using a real-life application.

Consider remote patient monitoring by which, with the help of wearable medical devices,

healthcare workers can monitor and capture medical and other health data from patients for assessment and, when necessary, treatment. A physician deploys a contract that includes a function “notify.” Let the time of deployment be  $t' < t$ , so that this contract appears as a row in the matrix  $T(t') = [L(t') \ S(t') \ R(t')]$ , where  $L(t')$  includes the patient’s information consisting of the patient history, information that results in the uniqueness of the device that is being used for monitoring the patient status, public keys that restrict the access to the contract to a particular group of people like trusted doctors and patient’s family, etc.  $S(t')$  contains pre-defined conditions that when met, the functions in  $R(t')$  including the function “notify,” must be called. While the wearable devices keep monitoring the patient’s health status, when a device detects an abnormality, it submits a transaction to the network, which corresponds to one row in matrix  $T(t) = [L(t) \ S(t) \ R(t)]$ . Here  $L(t)$  calls the deployed contract at epoch  $t'$  that corresponds to this patient.  $S(t)$  states the patient’s present status and that she needs to be hospitalized or be notified.  $R(t)$  contains the functions that need to be called for this purpose. It is the responsibility of the miners in the network to check the validity of the information in  $L(t)$  and  $S(t)$  with the corresponding existing information in the blockchain, which in our example is the related rows in  $L(t')$  and  $S(t')$ . In case of validity, the “notify” function is executed, letting the doctors know about the mentioned patient. The state  $S(t)$  will be updated and recorded in the blockchain.

After forming matrix  $T(t)$ , the BS sends this matrix to all miners in the network. Each miner checks the validity of the  $n$  contracts in  $T(t)$  based on the existing stored ledger in the previous blocks, i.e.,  $\mathbf{B}^{t-1} = \{B(1), \dots, B(t-1)\}$  where  $B(m) = [L^*(m) \ S^*(m) \ R^*(m)]$  consists of validated rows of  $L(m)$ ,  $S(m)$ , and  $R(m)$ , respectively. To check the validity of the contracts in  $T(t)$ , miner  $j$  computes  $v_j(t) = V_j^t(L(t), S(t), \mathbf{B}^{t-1}) \in \{0, 1\}^n$ , where  $V_j^t$  is the verification operator for miner  $j$  at epoch  $t$ , and 1/0 indicate whether or not a contract is valid. Next, each miner sends its verification result vector to other miners. All miners in the network apply a consensus algorithm to determine the global validity of the contracts by computing  $C(\{v_j(t)\}_{j=1}^N) = v^*(t) \in \{0, 1\}^n$ . For instance, under the majority rule,  $C$  selects the set of transactions that are declared valid by the majority of miners. Then, using  $v^*(t)$ ,  $L^*(t)$  and  $R^*(t)$  can be determined.

Each miner  $j$  executes the valid contracts by calling the functions in  $R^*(t)$ , i.e., it computes  $F_j^t(R^*(t), S(t)) = S_j(t)$ , where  $F_j^t$  is the function calling operator for miner  $j$ , that updates the state of the valid contracts to  $S_j(t)$ , which is then broadcasted to the network. Given the updated states from all miners, the confirmed updated state  $S^*(t)$  is obtained by computing  $U(\{S_j^t\}_{j=1}^{N(t)}) = S^*(t)$ , where similar to  $C$ ,  $U$  is an operator that is applied to the received updated states from all miners, and determines the updated states of transactions that are in consensus. The updated state  $S^*(t)$  along with  $L^*(t)$  and  $R^*(t)$  forms the block  $B(t) = [L^*(t) \quad S^*(t) \quad R^*(t)]$  for epoch  $t$ . Finally, each miner appends  $B(t)$  as the new block to its local copy of the blockchain  $\mathbf{B}^t = \mathbf{B}^{t-1} \cup B(t)$ .

The above conventional blockchain suffers from the so-called trilemma problem [52]–[54], i.e., the problem of being unable to balance simultaneously security, decentralization, and scalability, which means when one or two of these three metrics improve, the rest will be compromised. Coding techniques are one of the promising approaches to addressing the trilemma problem in the blockchain. Next, we will explain the mining process in the existing coded IoT blockchain.

### 2.2.3 Features of LCC-based Coded Blockchains

In addition to having low throughput, another drawback of the conventional consensus algorithms is that a small fraction of miners make the majority of network decisions [55]–[57]. In a coded blockchain, the goal is to have a decentralized high throughput scheme where all miners work together on computational tasks. We focus on the coded blockchain schemes LCB [20] and Polyshard [12] that employ the Lagrange Coded Computing (LCC). Although the two schemes apply the LCC in different ways, they share common features that are described below.

Suppose that the network wants to reach consensus about the validity of the data block  $X(t)$ . Each miner  $j$  first obtains the encoded block  $\tilde{X}_j(t) = E_j^t(X(t))$ , where  $E_j^t$  is the encoding operator for miner  $j$  which is a Lagrange interpolator at a specific point. Meanwhile, in a coded blockchain, instead of the raw data, miners store the encoded blocks of the previous epochs. Therefore, miner  $j$  has access to  $\tilde{\mathbf{B}}_j^{t-1} = \{\tilde{B}_j(1), \dots, \tilde{B}_j(t-1)\}$  which is its share of the coded blocks. Then miner  $j$  applies the operator  $H_j^t$  to the received data and computes  $H_j^t(\tilde{X}_j(t), \tilde{\mathbf{B}}_j^{t-1}) = \tilde{h}_j^t$  and broadcasts the

results to all other miners, where  $H_j^t$  is a verification function on coded data. Note that unlike the uncoded blockchain,  $\tilde{h}_j^t$  is not the verification results of miner  $j$ , but an intermediate coded result. Having the intermediate results from all miners, each miner  $j$  computes  $D^t(\{\tilde{h}_i^t\}_{i=1}^{N(t)})$ , where  $D^t$  is the decoding operator, to find the consensus decision on the validity of  $X(t)$ . An important feature of the LCC scheme is that miners can know the consensus decision even if some of the  $\tilde{h}_i^t$ 's are erroneous, i.e., the  $i$ -th miner is malicious, or are not received, i.e., the  $i$ -th miner is a straggler. Therefore, miners just need to wait for the intermediate results from a subset of the fastest honest miners to reach the consensus decision. Another feature is that miners store the ledger in coded form, and that the computation is performed on coded data. It should be noted that in Polyshard [12] LCC is used for the block verification stage only, while in LCB [20] LCC is used for both block generation and verification stages.

#### 2.2.4 Challenges in Uncoded and Coded IoT Blockchains

First, the number of transactions generated in IoT networks is typically significantly higher than that in conventional blockchains. The low throughput of conventional blockchains makes them highly inefficient. On the other hand, in the existing coded schemes, the Collateral Invalidation (CI) rate [44], which is the number of abandoned transactions due to one invalid transaction, is very high, making them inefficient for IoT blockchains.

Second, in the existing blockchains, the mechanism for faster processing is the higher transaction fees. However, in IoT applications, all transactions do not have the same priority due to their versatility. For example, transactions that control a health center or security of a sensitive place have higher priority than those in retail environments for automatic checkout. Therefore, the transaction selection and consensus procedure must be designed more carefully.

### 2.3 Proposed Coded IoT Blockchain

To address the above-mentioned shortcomings of the existing coded IoT blockchains, and to simultaneously achieve a high level of security, decentralization, and scalability, we propose a new

coded IoT blockchain that is based on our recent work of rateless coded blockchain [49]. The main new component of the proposed scheme is a transaction assignment stage designed to increase the throughput. In this section, we describe the rateless coded blockchain structure and the main steps of the mining procedure of the proposed coded IoT blockchain. Sections V will compare our proposed system with the existing coded IoT blockchains under different performance metrics.

### 2.3.1 Problem Statement

Throughput ( $\Theta(t)$ ) in blockchain quantifies the number of transactions processed and appended to the blockchain per epoch—a fixed interval during which transactions are collected, validated, and stored. Maximizing throughput involves selecting an optimal subset of transactions  $\mathcal{T}_s(t) \subseteq \mathcal{T}(t)$  from the incoming pool  $\mathcal{T}(t)$ , ensuring that resource constraints are met while minimizing computational waste. This optimization problem is expressed as:

$$\text{Maximize } \Theta(t) = \sum_{j \in \mathcal{T}_s(t)} \mathbf{1}_{\text{valid}(j)}, \quad (2.1)$$

$$\text{s.t. } \sum_{j \in \mathcal{T}_s(t)} \mathcal{P}_j \leq \mathcal{R}(t), \quad (2.2)$$

where  $\mathbf{1}_{\text{valid}(j)} = 1$  if transaction  $j$  is validated and successfully appended to the blockchain as part of a block, and  $\mathbf{1}_{\text{valid}(j)} = 0$  otherwise. Here,  $\mathcal{P}_j$  represents transaction  $j$ 's resource consumption, and  $\mathcal{R}(t)$  is the resource capacity available during the epoch.

To enhance throughput optimization, three key system constraints are defined in this chapter. The block size constraint ensures the total size of selected transactions does not exceed the encoded block's capacity ( $\mathcal{R}^{\text{size}}$ ), defined as  $\sum_{j \in \mathcal{T}_s(t)} \mathcal{P}_j^{\text{size}} \leq \mathcal{R}^{\text{size}}$ . This balances block utilization with storage efficiency. The computational capacity constraint governs the network's ability to encode, decode, and validate transactions, ensuring that  $\sum_{j \in \mathcal{T}_s(t)} \mathcal{P}_j^{\text{comp}} \leq \mathcal{R}^{\text{comp}}$ . This prevents processing delays and overloads, maintaining operational throughput despite the added complexity of coded data.

Additionally, validation depth constraints address the overhead introduced by accessing historical encoded data for transaction validation. Transactions with greater validation depth ( $\mathcal{P}_j^{\text{depth}}$ ) require more intensive decoding operations to retrieve relevant blocks that are buried deeper in blockchain. The system enforces a maximum validation depth ( $\mathcal{R}^{\text{depth}}$ ) to prioritize transactions with lower validation requirements, reducing computational burden and preserving system efficiency as setting  $\max_{j \in \mathcal{T}_s(t)} \mathcal{P}_j^{\text{depth}} \leq \mathcal{R}^{\text{depth}}$  ensuring encoded blockchain systems maintain high throughput despite the complexity of data retrieval.

Moreover, to transcend the goal of merely maximizing the quantity of processed transactions, the proposed framework incorporates a **transaction prioritization** mechanism. This mechanism shifts the objective from solely maximizing the number of transactions to maximizing the **cumulative reward**, denoted by  $\Theta_{\text{eff}}(t)$ , which accounts for the intrinsic value of each transaction based on factors such as urgency, fees, and age. Formally, the enhanced optimization problem considering the three key system constraints is defined as:

$$\text{Maximize } \Theta_{\text{eff}}(t) = \sum_{j \in \mathcal{T}_s(t)} r_j \mathbf{1}_{\text{valid}(j)}, \quad (2.3)$$

$$\text{s.t. } \sum_{j \in \mathcal{T}_s(t)} \mathcal{P}_j^{\text{size}} \leq \mathcal{R}^{\text{size}}, \quad (2.4)$$

$$\sum_{j \in \mathcal{T}_s(t)} \mathcal{P}_j^{\text{comp}} \leq \mathcal{R}^{\text{comp}}, \quad (2.5)$$

$$\max_{j \in \mathcal{T}_s(t)} \mathcal{P}_j^{\text{depth}} \leq \mathcal{R}^{\text{depth}}. \quad (2.6)$$

where in this refined formulation,  $r_j$  denotes the reward associated with transaction  $j$ , which reflects its utility based on vitality, transaction fees, and age.

Furthermore, efficient miner allocation enhances throughput by distributing computational tasks effectively, reducing validation time and delays, and optimizing resource use. This increases transaction validation capacity and enables the network to handle high volumes without compromising performance.

In the following, in Section IV-A, we discuss how urgency, fees, and age of transactions

are incorporated into  $\Theta_{\text{eff}}(t)$  as a reward maximization problem, followed by modeling problem (2.3)-(2.6) as a combinatorial optimization problem. Then, in Section IV-B, we address determining the optimal number of miners for processing the transactions.

### 2.3.2 Background on Rateless Coded Blockchain

First, we give a brief description of the encoding process employed in our blockchain. Recall that at the end of epoch  $t$  the network forms block  $B(t) = [L^*(t) \quad S^*(t) \quad R^*(t)]$  which consists of the verified rows of the matrix of received transactions  $T(t) = [L(t) \quad S(t) \quad R(t)]$ . Encoding is performed every  $W$  epochs on  $W$  consecutive blocks. That is, at epoch  $t = \ell W$ , each miner  $j$  encodes blocks  $B((\ell - 1)W + 1), \dots, B(\ell W)$  into a coded block  $\mathbf{c}_j(\ell)$  as follows. First, we represent each data block  $B((\ell - 1)W + w)$  with a column vector  $\mathbf{b}_\ell(w)$  of size  $s$  over the finite field  $\mathbb{F}_q$ , where  $q = 2^p$  and  $p$  is the number of bits in each elements of  $\mathbf{b}_\ell(w)$ ,  $w = 1, \dots, W$ . Then, a systematic block code with generator  $\mathbf{G}$  of size  $W \times \bar{W}$  is applied to obtain  $[\mathbf{b}_\ell(1), \dots, \mathbf{b}_\ell(W)]\mathbf{G} = [\mathbf{d}_\ell(1), \dots, \mathbf{d}_\ell(\bar{W})]$ , where  $\bar{W} > W$  and  $\{\mathbf{d}_\ell(w), w = 1, \dots, \bar{W}\}$  are intermediate codewords or pre-codewords. Since  $\mathbf{G}$  is systematic, we have  $\mathbf{d}_\ell(i) = \mathbf{b}_\ell(i)$  for  $i = 1, \dots, W$ . Next each miner  $j$  further encodes the  $\bar{W}$  intermediate codewords into a coded block  $\mathbf{c}_\ell(j)$  using a systematic LT code. Specifically, for  $j = 1, \dots, \bar{W}$ , miner  $j$  simply stores  $\mathbf{d}_\ell(j) = \mathbf{c}_\ell(j)$ , corresponding to the information part of the code. All other miners store the parity part of the code. In particular, each miner  $j$ ,  $j = \bar{W} + 1, \dots, N(\ell W)$ , first chooses a degree  $L_j$  from a generator degree distribution  $\Omega(L)$ . It then randomly chooses  $L_j$  intermediate codewords  $\{\mathbf{d}_\ell(i_1), \dots, \mathbf{d}_\ell(i_{L_j})\}$  from  $\{\mathbf{d}_\ell(1), \dots, \mathbf{d}_\ell(\bar{W})\}$  and computes  $\mathbf{c}_\ell(j) = \mathbf{d}_\ell(i_1) \oplus \dots \oplus \mathbf{d}_\ell(i_{L_j})$ . The coded block  $\mathbf{c}_\ell(j)$  is stored by miner  $j$ . Recall that  $N(\ell W)$  is the number of miners in the network at the time of encoding  $t = \ell W$ . We choose  $\bar{W}$  such that  $N(\ell W) > \bar{W} > W$ . The encoding scheme described above is shown in Fig. 1.

In our scheme miners process transactions using the raw data. Suppose miner  $j$  needs access to a subset of blocks, denoted by  $\hat{\mathbf{B}}_j(\ell)$ , from the group of blocks  $\{B((\ell - 1)W + 1), \dots, B(\ell W)\}$ , i.e., some of the systematic code words in  $\{\mathbf{b}_\ell(1), \dots, \mathbf{b}_\ell(W)\}$ ,  $\{\mathbf{d}_\ell(1), \dots, \mathbf{d}_\ell(W)\}$ , or  $\{\mathbf{c}_\ell(1), \dots, \mathbf{c}_\ell(W)\}$ . Now, to obtain the required data, there are two approaches. First, miner  $j$  can repair the corresponding

intermediate codewords using the RNM algorithm, which is described in detail in Appendix, by exploiting the bitwise xor relationship between  $\mathbf{c}_{\ell(i)}$ 's and  $\mathbf{d}_{\ell(i)}$ 's. This essentially corresponds to decoding the outer LT codes. But instead of decoding all  $\overline{W}$  intermediate codewords, the RNM algorithm decodes only the required intermediate blocks. Second, in case the first approach cannot obtain all required blocks, then a full decoding of both the outer LT code and inner block code needs to be carried out to obtain the group of original data blocks  $\{\mathbf{b}_{\ell(1)}, \dots, \mathbf{b}_{\ell(W)}\}$ . Note that for repairing, the number of involved blocks is typically proportional to the generated degree  $L_j$ , while for decoding this number becomes proportional to the number of miners  $N(\ell W)$ . Since  $N(\ell W) \gg L_j$ , repairing is computationally much more efficient than decoding. Fortunately, most of the time only repairing is needed for a miner to obtain its required data and decoding is rarely needed. The details of the RNM algorithm and the decoding procedure can be found in [49], [58] and references therein.

Finally, we briefly discuss the choice of the size  $W$  of the block group. On the one hand, since for every  $W$  data block, each miner stores only one coded block,  $W$  represents the compression ratio and should be large. On the other hand, if  $W$  is too large and some miners may leave the network, this could result in unsuccessful decoding, and miners may not be able to recover some data blocks during the time interval between encoding instants  $(\ell - 1)W$  and  $\ell W$ . Therefore, there is a trade-off between the decoding failure probability and the reduction in storage. The method for finding the optimum  $W$  is discussed in [49] by using analysis tools in [58], [59].

### 2.3.3 Overview of the Mining Process

The mining process in our approach goes beyond the conventional task of merely generating blocks; it's a carefully structured, multi-stage mechanism tailored to the unique demands of IoT blockchains. Traditional mining typically involves validating transactions or solving cryptographic puzzles to create blocks. However, IoT networks introduce additional complexities, such as managing high transaction volumes, optimizing limited computational and storage resources, and handling constantly changing network topologies.

Our enhanced mining process, therefore, is designed to address these specific challenges with four

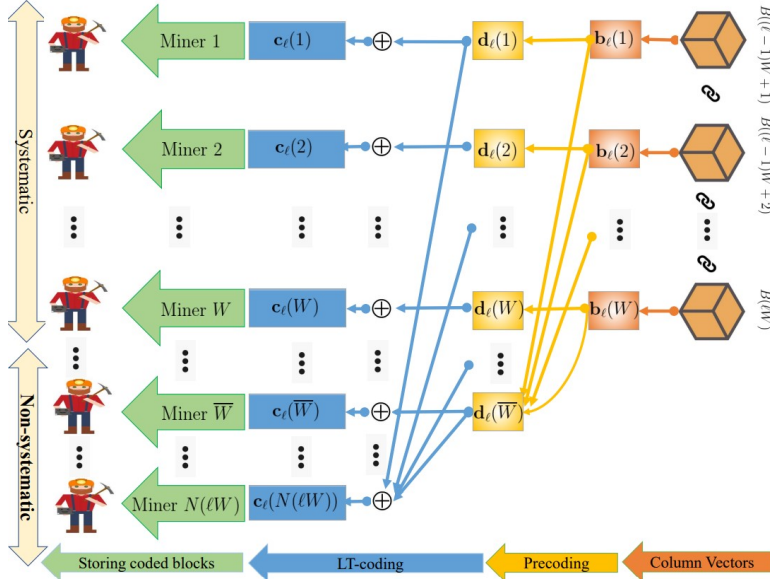


Figure 2.1: Block coding in the proposed scheme using a systematic raptor code.

interconnected stages: transaction selection, miner assignment, transaction and block verification, and block encoding for storage efficiency. Each stage contributes distinct, targeted functionalities that improve scalability, resource optimization, and security within the IoT environment.

The process begins with selective transaction handling, where the BS prioritizes transactions based on factors like urgency, transaction fees, and age. This approach ensures efficient processing of high-priority transactions despite limited resources, as detailed in Section IV-A. Next, the BS assigns these prioritized transactions to a subset of reliable miners, carefully distributing the workload to prevent resource overload and foster scalability, as described in Section IV-B.

Following assignment, the miners verify transactions in a decentralized manner, ensuring that only valid transactions are added to the blockchain. This stage, crucial for maintaining blockchain integrity, also includes consensus formation and state updates, detailed further in Sections III-D and IV.

Finally, to address IoT storage constraints, we employ rateless Raptor codes to encode blocks, enabling miners to store blockchain data efficiently. This encoding approach allows each miner to retain only partial data fragments, which can later be reconstructed if needed, facilitating participation from storage-limited IoT devices as this strategy explored in Sections III-B and III-D.

By integrating these stages, our mining process extends beyond block generation to offer a robust framework optimized for the dynamic and resource-constrained nature of IoT networks. This structured approach ensures scalability, adaptability, and secure data handling within an IoT blockchain ecosystem.

#### 2.3.4 Proposed Mining Procedure

In our proposed scheme, the BS is assumed to be a trusted entity that is always available. The BS collects transactions, performs transaction assignment, and selects miners. This entity is critical for coordinating the workflow but does not participate in mining or consensus itself.

The blockchain operates as a permissioned blockchain where all participating miners know one another. In this setting, each miner is aware of the identities of other miners in the network. Moreover, miners also know the set of assigned miners for each transaction batch. This transparency ensures that, as described in Section IV.B, every miner can collect and verify votes from other miners, facilitating a reliable and verifiable consensus process.

Our proposed mining scheme consists of four stages, as described in the following.

**Stage 1: Transaction Assignment:** At the beginning of epoch  $t$  the BS collects  $n$  transactions from different devices to form the matrix  $T(t) = [L(t) \ S(t) \ R(t)]$ . Next, the BS performs transaction selection and miner assignment, as follows.

- (i) Transaction selection: By solving an optimization problem, the BS selects  $K(t)$  of the  $n$  transactions that are more likely to be appended to the blockchain. By doing so, we try to minimize the amount of futile work in the network and speed up the verification process. Therefore, this will increase the throughput and lead to a shorter confirmation time for transactions, which is one of the most desired features in IoT blockchains.
- (ii) Miner assignment: Based on the current network conditions, the BS will determine the number of the miners,  $M(t)$ , each selected transaction must be assigned to, such that the network can reach a consensus about the validity of each transaction with high probability. Then the BS will assign each selected transaction to  $M(t)$  randomly selected miners.

As the result of the above two steps, the BS finds the set of selected transactions  $\mathcal{T}_s(t) = \{x_1, \dots, x_{K(t)}\}$  corresponding to  $K(t)$  rows of matrix  $T(t)$ , with the reduced matrix of the transactions  $\hat{T}(t) = [\hat{L}(t) \quad \hat{S}(t) \quad \hat{R}(t)]$ . Moreover, for each selected transaction  $x_i, i = 1, \dots, K(t)$ , a subset  $\mathcal{M}_i$  of miners with  $|\mathcal{M}_i| = M(t)$  is assigned for validating and further processing. Let  $q_j(t)$  be the number of transactions assigned to miner  $j$ . Then on average each miner is assigned  $q(t) = \frac{1}{N(t)} \sum_{j=1}^{N(t)} q_j(t) = K(t)M(t)/N(t)$  transactions to process. The details of the above two steps in Stage 1 are described in Section IV.

**Stage 2: Transactions Verification:** The BS sends to each miner the matrix of the assigned transactions  $\hat{T}_j(t) = [\hat{L}_j(t) \quad \hat{S}_j(t) \quad \hat{R}_j(t)]$  consisting of  $q_j(t)$  rows (transactions) from the reduced matrix of the transactions  $\hat{T}(t)$ . Using the notations in Section III-B, to check the validity of the assigned transactions, miner  $j$  needs some blocks  $\hat{\mathbf{B}}_j(\ell)$  from some of the past block groups  $\ell$ . Denote the set of all such required blocks  $\hat{\mathbf{B}}_j(\ell)$  for verification at epoch  $t$  by  $\hat{\mathbf{B}}_j^{t-1}$ . Recall that each miner stores a fraction of the coded data and uses it for verifying the assigned transactions. If some required data in  $\hat{\mathbf{B}}_j^{t-1}$  are not locally stored by miner  $j$ , it contacts a subset of the miners in the network and decodes the intermediate blocks corresponding to the missing data in  $\hat{\mathbf{B}}_j^{t-1}$ . If such a repair process fails to recover  $\hat{\mathbf{B}}_j^{t-1}$ , miner  $j$  then decodes all entire block groups that are associated with  $\hat{\mathbf{B}}_j^{t-1}$ .

Then having the required data  $\hat{\mathbf{B}}_j^{t-1}$ , miner  $j$  computes  $V_j^t(\hat{L}_j(t), \hat{S}_j(t), \hat{\mathbf{B}}_j^{t-1})$  to find the vector  $v_j^t \in \{0, 1\}^{q_j(t)}$  which is the verification results of miner  $j$ . Be noted that here the length of vector  $v_j^t$  is  $q_j(t)$ , so each transaction is not verified by all miners. Then each miner sends the vector  $v_j^t$  to other miners and the BS. Upon receiving the validation vectors from all miners, based on the miner assignment  $\mathcal{M}(t) = \{\mathcal{M}_1, \dots, \mathcal{M}_{K(t)}\}$  for the  $K(t)$  selected transactions, each miner checks if for each transaction  $x_i$ , there are enough miners in  $\mathcal{M}_i$  in favor of its validity, i.e., the majority of the miners in  $\mathcal{M}_i$  accept  $x_i$  as a valid transaction. In case of majority acceptance,  $x_i$  is validated. As a result of the mentioned procedure, miners in the network can reach a consensus about the set of valid transactions  $\hat{L}^*(t)$ .

*Remark 1:* Denote  $d_i$  as the depth of the oldest block that is required for processing transaction

$x_i$ . That is if  $x_i$  is generated at epoch  $t$ , then the oldest block required for processing  $x_i$  is generated at epoch  $t - d_i$ . In the Transaction Selection step (see Section IV-A), the BS limits the depth of selected transactions, such that  $d_i < D$ . For example, for a batch of incoming transactions at epoch  $t$ , where  $\ell W < t \leq (\ell + 1)W$ , if  $D = t - \ell W$ , then the BS selects transactions whose validations require blocks generated during the time interval  $[\ell W, t]$ , which are not encoded yet and are locally stored. All other transactions will be backlogged. On the other hand, if  $D = t - (\ell - 1)W$ , then the required old blocks are generated during  $[(\ell - 1)W, t)$ , which involves the most recently encoded blocks. And so on. Hence the BS can adjust  $D$  based on the number of backlogged transactions and the new incoming transactions in each epoch so that the raw data required for validation can be successfully obtained through repairing, keeping the decoding complexity low.

*Remark 2:* Sharding where a set of miners are partitioned into disjoint groups, and each group verifies a set of disjoint transactions, is a particular case of the mentioned scheme. In our scheme, the sets of miners for different transactions could be disjoint or have some members in common.

*Remark 3:* The RNM algorithm employed here is the same as that in [49] with a difference in usage. There the algorithm is used by newly joined miners only to store the ledger by recovering the previous blocks. On the other hand, in our scheme, in addition to the mentioned purpose, it is used to recover the blocks if they are required for transaction verification.

*Remark 4:* In Stage 2, instead of broadcasting raw vote vectors  $v_j^t$ , miners attach cryptographic hash commitments (e.g., using SHA-256) to their votes. This approach binds each miner to a unique vote value, so any attempt at equivocation, such as sending different votes to the BS versus other miners, would lead to mismatched hash values detectable by the network. Consequently, these cryptographic commitments ensure consistency and integrity of votes, preventing malicious discrepancies and preserving the reliability of the consensus process.

**Stage 3: Block Verification:** At this step, each miner  $j$  uses the set of valid transactions  $\hat{L}^*(t)$  and forms the set  $\hat{L}_j^*(t)$ , which is a subset of the confirmed transactions from  $\hat{L}^*(t)$ , where the transactions that are not in  $\hat{T}_j(t)$  are excluded. Denoting  $\hat{R}_j^*(t)$  as the corresponding functions that are required to be called from the set  $\hat{L}_j^*(t)$ , miner  $j$  finds the updated state of  $\hat{S}_j(t)$  for its assigned

contracts by computing  $F_j^t(\hat{R}_j^*(t), \hat{S}_j(t)) = \hat{S}_j(t)$ , where  $F_j^t$  is the function calling operator for miner  $j$ . The updated state will be broadcasted to the network and the BS. Similar to the previous stage, if the majority of the miners in  $\mathcal{M}_i$  broadcasted the same updated state for transaction  $x_i$ , that state becomes the updated state for  $x_i$ . This way, the updated state  $\hat{S}^*(t)$  for all confirmed transactions at Stage 2 are determined. Each miner appends the block  $\hat{B}(t) = [\hat{L}^*(t) \quad \hat{S}^*(t) \quad \hat{R}^*(t)]$  to its local storage until the moment of the group block encoding that is explained in the next stage. Also, the BS updates the status of the network and reliability of each miner by comparing the received results of computations with the collective decisions on the validity of transactions which will be detailed in Section IV-B.

**Stage 4: Block Encoding:** Recall that in our scheme the blocks are encoded in groups of size  $W$ . Suppose that  $\hat{B}((\ell - 1)W + 1)$  is the first block in the  $\ell$ -th group of blocks. Stages 1-3 repeat for all epochs  $t = (\ell - 1)W + 1$  through  $t = \ell W$ . After the generation of the block  $\hat{B}(\ell W)$ , it is the time for encoding the  $\ell$ -th block group, which are blocks  $\{\hat{B}((\ell - 1)W + 1), \hat{B}((\ell - 1)W + 2), \dots, \hat{B}(\ell W)\}$ . The group size  $W$  is determined by the BS and based on the number of active nodes in the network. First, as the pre-coding step, miners in the network encode blocks  $\{\mathbf{b}_\ell(1), \dots, \mathbf{b}_\ell(W)\}$  to form the intermediate codewords  $\{\hat{\mathbf{d}}_\ell(1), \dots, \hat{\mathbf{d}}_\ell(\bar{W})\}$ . Then following the procedure in Section III-B each miner  $j$  generates and stores one coded block  $\mathbf{c}_\ell(j)$ . Afterward, miners erase all the original blocks  $\{\mathbf{b}_\ell(1), \dots, \mathbf{b}_\ell(W)\}$ . But, even though it is not necessary, depending on the storage capability of each miner, they could keep the intermediate blocks  $\{\hat{\mathbf{d}}_\ell(1), \dots, \hat{\mathbf{d}}_\ell(\bar{W})\}$  for some time, so that when these blocks are required for transactions verification, they are directly available without repairing the corresponding coded block. Hence it is an instance of trading storage for computing.

It is worth mentioning that in any stage during the generation of the blocks  $\{\hat{B}(i)\}_{i=(\ell-1)W+1}^{\ell W}$  if miners need any data from blocks that are older than  $\hat{B}((\ell - 1)W + 1)$ , they need to contact other miners and perform the repair process or decoding to obtain the data. However, if the required data is in the blocks  $\{\hat{B}(i)\}_{i=(\ell-1)W+1}^{i=\ell W}$  because the encoding of the  $\ell$ -th group of blocks has yet to be started, that data are stored locally by each miner as raw data.

Also, note that due to the use of rateless codes, given the set of data blocks in one group

as the source symbols, one can generate potentially unlimited number of encoded symbols. Specifically, each newly joined miner  $j'$  can generate and store  $\mathbf{c}_\ell(j')$  by first generating the degree number  $L_{j'}$  from degree distribution  $\Omega(L)$  and then selecting  $L_{j'}$  intermediate blocks from the set  $\{\hat{\mathbf{d}}_\ell(1), \dots, \hat{\mathbf{d}}_\ell(\overline{W})\}$  to form its coded block. Therefore, our scheme adapts to dynamic networks, and miners can join and leave the network without having an adverse effect on the encoding and decoding procedures. Such a feature is especially well suited for IoT networks which are typically highly dynamic. However, other coding schemes, such as the LCC-based coded blockchain, are designed for a network with a fixed number of nodes.

Furthermore, the reason for choosing the raptor codes as the coding scheme for our blockchain, in addition to being a linear time encoding and decoding scheme, is that it has a very low decoding failure probability, i.e., the probability that a miner fails to decode a group of blocks is very low, which makes it a suitable candidate for IoT blockchains with strict security and timing requirements.

The proposed mining procedure for the coded IoT blockchain is detailed in Algorithm 1, outlining the sequential stages of transaction assignment, verification, consensus, and block encoding.

## 2.4 Transaction Assignment

In this section, we describe the transaction selection and the miner assignment steps in the first stage of our proposed mining procedure for coded IoT blockchain.

### 2.4.1 Transaction Selection

Our goal is to select a subset of  $n$  transactions to maximize the total reward subject to constraints on computing and storage resource as well as the depth of required blocks for processing transactions. Three factors determine the reward of a transaction:

1. *Vitality*: It is an indicator of the level of urgency for the transaction. A higher value means the transactions should be processed faster.
2. *Transaction fee*: With the same vitality level, it is preferred to first work on transactions with

---

**Algorithm 1: Coded IoT Blockchain Mining Procedure**

---

**Stage 1: Transaction Assignment**Collect  $n$  transactions  $T(t) = [L(t), S(t), R(t)]$ 

Assign transactions and miners (Refer to Algorithm 2)

**Stage 2: Transaction Verification****foreach** miner  $j$  assigned transactions  $\hat{T}_j(t)$  **do**    Retrieve required prior blocks  $\hat{\mathbf{B}}_j^{t-1}$     Verify transactions:  $v_j^t = V_j^t(\hat{L}_j(t), \hat{S}_j(t), \hat{\mathbf{B}}_j^{t-1})$     Broadcast verification result  $v_j^t$  to other miners and the Base Station (BS)**end****Consensus:** Finalize valid transactions  $\hat{L}^*(t)$ **Stage 3: Block Verification****foreach** miner  $j$  **do**    Execute valid contracts:  $\hat{S}_j(t) = F_j^t(\hat{R}_j^*(t), \hat{S}_j(t))$     Broadcast updated state  $\hat{S}_j(t)$ **end****Consensus:** Update global state  $\hat{S}^*(t)$ Form block:  $\hat{B}(t) = [\hat{L}^*(t), \hat{S}^*(t), \hat{R}^*(t)]$ **Stage 4: Block Encoding****if**  $t = \ell W$  **then**    Encode blocks  $\{\hat{B}((\ell - 1)W + 1), \dots, \hat{B}(\ell W)\}$     Store encoded block  $\mathbf{c}_\ell(j)$  for each miner    Discard raw blocks  $\{\hat{B}((\ell - 1)W + 1), \dots, \hat{B}(\ell W)\}$ **end**

---

higher fees to maximize the profit of the miners.

3. *Age*: It is the time between the submission of the transaction and sending the transaction to the miners for processing. If a transaction does not have a high priority or a competitive transaction fee, it should not be prevented from being sent for processing because many backlogs can result in network congestion.

We use Richards' curve [60] to model the reward of each transaction based on the above three factors. Denote  $v_j$ ,  $a_j$ , and  $f_j$  as the vitality, age of the transactions in number in epochs, and transaction fee, respectively, of the  $j$ -th transaction,  $j = 1, \dots, n$ . Denoting  $\mathbf{v} = (v_1, \dots, v_n)$ , and  $\mathbf{a} = (a_1, \dots, a_n)$ ,

the reward of the  $j$ -th transaction is given by:

$$r_j(\mathbf{v}, \mathbf{a}, f_j) = \frac{1}{(1 + \tilde{v}_j e^{-f_j})^{1/\tilde{a}_j}}, \quad (2.7)$$

where

$$\tilde{v}_j = \frac{e^{-\frac{1}{\|\mathbf{v}\|} v_j}}{\sum_{k=1}^n e^{-\frac{1}{\|\mathbf{v}\|} v_k}}, \quad \tilde{a}_j = \frac{e^{\frac{1}{\|\mathbf{a}\|} a_j}}{\sum_{k=1}^n e^{\frac{1}{\|\mathbf{a}\|} a_k}}. \quad (2.8)$$

Fig. 2 shows the efficiency of the proposed model for transaction prioritization. For  $n = 5$  with  $\mathbf{v} = (1, 10, 6, 8, 2, 4)$  and  $\mathbf{a} = (16, 1, 4, 32, 2, 8)$ , we plot  $r_j$  as a function of  $f_j$  for  $j = 1, 2, 3$ . From the figure we can see in the high fee region, e.g.,  $f_j > 10$ , the transaction with high vitality ( $j = 2$ ) has higher priority and approaches the maximum faster. On the other hand, in the lower fee region, its priority becomes lower than the transaction with older age ( $j = 1$ ).

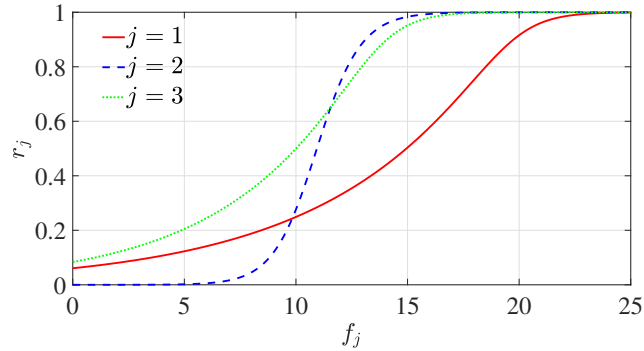


Figure 2.2: Plot of the reward function  $r_j(\mathbf{v}, \mathbf{a}, f_j)$ .

For each transaction  $j$ , define  $\xi_j$  as the amount of computation resource required for processing it, modeled as a Gamma random variable with the shape parameter  $\alpha_j$  and scale parameter  $\beta$ ; define  $\eta_j$  as its size, modeled as a Gaussian random variable with mean  $\mu_j$  and variance  $\sigma^2$ ; and define  $d_j$  as the maximum depth, which is the position of the oldest block required for processing transaction  $j$ , modeled as a Poisson random variable with mean  $\lambda_j$ . Furthermore, define the binary variable  $x_j = 1$  if the  $j$ -th transaction is selected for processing and  $x_j = 0$  otherwise. Then the problem of

selecting a subset of transactions can be formulated as the following optimization problem:

$$\max_{x_j \in \{0,1\}, j=1, \dots, n} \sum_{j=1}^n r_j x_j, \quad (2.9a)$$

$$s.t. \mathbb{P} \left( \sum_{j=1}^n \xi_j x_j \leq C \right) = P \left( \sum_{j=1}^n \alpha_j x_j, \frac{C}{\theta} \right) \geq q_1, \quad (2.9b)$$

$$\mathbb{P} \left( \sum_{j=1}^n \eta_j x_j \leq S \right) = \Phi \left( \frac{S - \sum_{j=1}^n \tau_j x_j}{\sqrt{\omega \sum_{j=1}^n \tau_j x_j}} \right) \geq q_2, \quad (2.9c)$$

$$\mathbb{P} (\max(d_1 x_1, \dots, d_n x_n) \leq D) = \prod_{j=1}^n Q(D + 1, \lambda_j x_j) \geq q_3. \quad (2.9d)$$

where  $P$ ,  $\Phi$ , and  $Q$  are the lower regularized gamma function, the cdf of standard normal and the upper regularized gamma function, respectively. Formulation (2.9) maximizes the total reward of the selected transactions subject to three probabilistic constraints: (2.9b) puts a constraint on the computational cost for the selected transactions; (2.9c) limits the aggregated size of the selected transactions to the size of one block; and (2.9d) restricts the depth of the required blocks for processing the selected transactions.

Since in an IoT blockchain, the number of transactions is typically much higher than that in conventional blockchains like cryptocurrency blockchains, the traditional mining process in which each miner processes all incoming transactions becomes inefficient. Here we cluster the incoming transactions for faster and more efficient processing leading to higher throughput. First, based on the transactions' type and calling functions, they would consume a wide range of computing resources; hence, they need to be judiciously chosen to meet the computing capacity limit. Second, due to the diversity of the transactions and contracts in the IoT blockchain, the size of each transaction could be in a broader range than conventional blockchains. Hence if a transaction is not likely to be accommodated in a block along with other submitted transactions, it will not be selected. Third, by restricting the depth of selected transactions to only a few previous blocks, in case miners could not repair the required blocks and had to decode a group, we limit the total decoding complexity.

Alternatively, when  $\xi_j$ ,  $\eta_j$ , and  $d_j$  are known to the BS, we can also consider the following

deterministic optimization formulation:

$$\max_{x_j \in \{0,1\}, j=1, \dots, n} \sum_{j=1}^n r_j x_j, \quad (2.10a)$$

$$s.t. : \sum_{j=1}^n \xi_j x_j \leq C, \quad (2.10b)$$

$$\sum_{j=1}^n \eta_j x_j \leq S, \quad (2.10c)$$

$$\max(d_1 x_1, \dots, d_n x_n) \leq D. \quad (2.10d)$$

Problems (2.9) and (2.10) are stochastic and deterministic forms of the well-known knapsack problem [61]–[63] in combinatorial optimization, respectively, which are NP-complete [64]. We relax them by changing the constraints  $x_j \in \{0, 1\}$  to  $x_j \in [0, 1]$ . It is shown in [65] that the stochastic knapsack problem in (2.9) is a convex optimization problem in its relaxed form. Moreover, the relaxed version of the deterministic problem (2.10) is a linear program. Finally, the solution to the relaxed problem can be mapped to a binary solution by using the random rounding method [66].

Note that the stochastic formulation (2.9) and the deterministic one in (2.10) have their own advantages and drawbacks. Formulation (2.9) is more demanding computationally, but the BS does not need to know the exact values of different transaction attributes, i.e., size, amount of required resource for computation, and depth of the oldest required block for verification. On the other hand, though formulation (2.10) is less demanding computationally, the BS must know the attribute values of each transaction which may not be available for some IoT blockchains.

#### 2.4.2 Miner Assignment

In our scheme, the BS keeps monitoring the reliability of each miner. Specifically, denote  $p_j(t)$  as the probability that miner  $j$  is reliable, estimated by the BS at epoch  $t$ . Recall that  $v^*(t) \in \{0, 1\}^{K(t)}$  is the binary vector whose  $i$ -th element is 1 if the majority of miners validate the  $i$ -th selected transaction by the BS, and 0 otherwise. Also,  $v_j(t) \in \{0, 1\}^{q_j(t)}$  contains votes of

miner  $j$  at epoch  $t$  about the validity of its set of assigned transactions. Denote  $v_j^*(t) \in \{0, 1\}^{q_j(t)}$  as the subvector of  $v^*(t)$  that includes just the corresponding elements of  $v_j(t)$ . Now assume that at time epoch  $t - 1$ , the reliability of miner  $j$  is estimated as  $p_j(t - 1)$ . Then after receiving all votes from all miners in the network and knowing the collective decision about the validity of each transaction, the BS can update the reliability of miner  $j$  as follows:

$$p_j(t) = (1 - \beta)p_j(t - 1) + \beta \frac{l_j(t)}{q_j(t)} \quad (2.11)$$

where  $0 < \beta < 1$  is a forgetting factor to count for time-evolution, and  $l_j(t)$  correct votes of miner  $j$  at epoch  $t$ , i.e., the number of zeros in  $v_j^*(t) \oplus v_j(t)$ . Now, suppose that at the beginning of epoch  $t$  the BS wants to assign each transaction from  $\hat{T}(t)$  to  $M(t)$  miners in the network in a way that after each miner collects the votes of other miners, the network is able to reach a consensus about the validity of transactions, based on the majority rule, with a probability of at least  $1 - \epsilon$  for  $0 < \epsilon < 1$ . Denote  $Z_j(t) = 1$  if miner  $j$ 's vote is reliable and 0 otherwise. Now denote  $Z(t) = Z_1(t) + \dots + Z_{M(t)}(t)$ . The network will make a correct decision if  $Z(t) > \frac{M(t)}{2}$ . Approximating the miners' votes as i.i.d. Bernoulli random variables with parameter  $P(t) = \left( \prod_{j=1}^{N(t)} p_j(t) \right)^{1/N(t)}$ , and using the Chernoff bound [67] we have:

$$\Pr \left[ Z(t) > \frac{M(t)}{2} \right] \geq 1 - e^{-\frac{1}{2P(t)} M(t) (P(t) - \frac{1}{2})^2} \geq 1 - \epsilon, \quad (2.12)$$

which results in:

$$M(t) = \left\lceil \frac{8(\ln 1/\epsilon)P(t)}{(1 - 2P(t))^2} \right\rceil \quad (2.13)$$

as the minimum number of miners required to be assigned to each transaction. Therefore, the BS computes  $P(t)$  and  $M(t)$  at the beginning of each epoch  $t$  and randomly assigns each of the  $K(t)$  selected transactions from the reduced matrix of transactions  $\hat{T}(t)$  to  $M(t)$  miners.

The process of transaction selection and miner assignment is described in Algorithm 2, where transactions are evaluated based on a reward function, optimized under resource constraints, and

assigned to miners.

---

**Algorithm 2:** Transaction Selection and Miner Assignment

---

**Input:** Transactions  $\{1, 2, \dots, n\}$  with vitality  $v_j$ , age  $a_j$ , fee  $f_j$ , resource requirements  $\xi_j$ , sizes  $\eta_j$ , depths  $d_j$

**Compute Rewards:**

**foreach** transaction  $j$  **do**

    | Compute reward  $r_j$  using equation (2.7)

**end**

**Formulate and Solve Optimization Problem:**

Maximize total reward  $\sum_{j=1}^n r_j x_j$  subject to constraints in equations (2.9b), (2.9c), and (2.9d)

Solve for  $x_j^*$

**Select** transactions where  $x_j^* = 1$  to form the set  $\mathcal{T}_s(t)$

**Compute** average miner reliability  $P(t)$  using equation (2.11)

**Determine** number of miners per transaction  $M(t)$  using equation (2.13)

**foreach** transaction  $x_i \in \mathcal{T}_s(t)$  **do**

    | Randomly assign transaction  $x_i$  to  $M(t)$  miners to form  $\mathcal{M}_i$

**end**

---

## 2.5 Comparison with Existing Coded Blockchains

In this section, we will compare our proposed scheme with two other benchmark papers on coded blockchains, which are Polyshard [12] and LCB [20] in terms of several performance metrics, including storage reduction, decentralization, throughput, and security. Although both papers use the Lagrange polynomial to code the blockchain data, Polyshard uses a sharded blockchain, while LCB is based on a non-sharded approach. The reason for choosing these two papers is that to the best of our knowledge, at the time of writing this chapter, these two works are the only benchmarks on coded blockchains that fit IoT applications.

### 2.5.1 Simulation Setup

We analyze a dynamic network where miners join and leave at the transition between epochs. The number of miners joining or leaving follows Poisson distributions with means  $\lambda_e$  and  $\lambda_l$ , respectively. In the pre-code stage, we employ the Reed-Solomon code from the Matlab 2021a Communication

Toolbox. The size of encoded groups in our scheme,  $W_\ell$ , is dynamic and varies for each group based on the number of miners in the network at encoding time.

The  $\ell$ -th group of encoded blocks consists of  $\{\hat{B}(W_{(\ell-1)} + 1), \dots, \hat{B}(W_\ell)\}$ , and its size is determined to maintain a pre-code rate of  $W_\ell/\bar{W} = 0.8$ . To calculate  $W_\ell$ , we use the failure probability function from [49], which estimates the probability of failing to decode at least one input symbol for a rateless code given its generator matrix  $\mathbf{G}$  and degree distribution  $\Omega(L)$ . By determining the overhead required to ensure reliable decoding with high probability and using the current number of miners  $N(t)$  (equal to the number of output coded symbols), we compute  $W_\ell$  such that the failure probability remains below a predefined threshold. The modified robust soliton distribution [68] is used as the degree distribution given by:

$$\Omega(L) = \begin{cases} 0, & \text{for } L = 1 \\ \mu(L) + \frac{\mu(1)}{\bar{W}-1}, & \text{for } L = 2, \dots, \bar{W} \end{cases} \quad (2.14)$$

where

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta}, \quad \text{for } 1 \leq i \leq \bar{W} \quad (2.15)$$

and  $\beta = \sum_{i=1}^{\bar{W}} (\rho(i) + \tau(i))$ . In (2.15),  $\rho(i)$  and  $\tau(i)$  are given by:

$$\rho(i) = \begin{cases} 1/\bar{W}, & \text{for } i = 1, \\ 1/(i(i-1)), & \text{for } 2 \leq i \leq \bar{W}, \end{cases} \quad (2.16)$$

$$\tau(i) = \begin{cases} \mathcal{S}/i\bar{W}, & \text{for } 1 \leq i \leq \frac{\bar{W}}{\mathcal{S}} - 1, \\ \mathcal{S} \ln(\mathcal{S}/\delta)/\bar{W}, & \text{for } i = \frac{\bar{W}}{\mathcal{S}}, \\ 0, & \text{otherwise.} \end{cases}$$

where  $\delta \in [0, 1]$  and  $c > 0$  are constant, and  $\mathcal{S} = c \cdot \sqrt{\bar{W}} \cdot \ln\left(\frac{\bar{W}}{\delta}\right)$  is the average number of degree-1 code symbols. We set  $c = 0.15$  and  $\delta = 0.5$ . As seen in Fig. 3, this degree distribution favors lower degrees more than higher degrees, which helps reduce the decoding complexity.

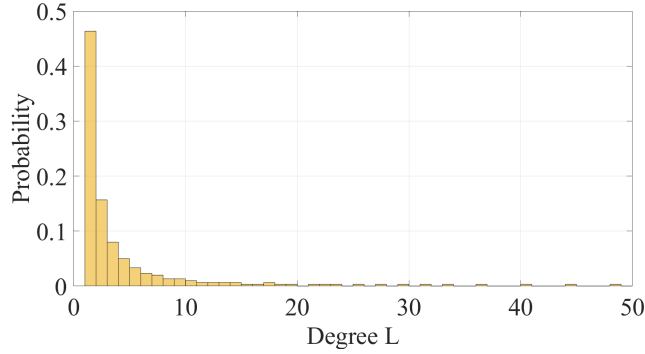


Figure 2.3: The modified robust soliton distribution used in the LT code ( $\bar{W} = 1000$ ,  $\delta = 0.15$ ,  $c = 0.5$ ).

Our simulation has three types of miners: dishonest, honest, and straggler. A dishonest miner always votes incorrectly, while an honest miner always successfully delivers correct votes to the network. Also, a straggler is a miner that, with probabilities of  $1/3$  and  $2/3$ , does not send any votes and sends correct votes to other miners, respectively. To have a fair comparison with LCB and Polyshard, adopting the parameters in [20], we assume that in all three schemes, up to 40% of miners in the network can be a straggler.

We distinguished straggler nodes from malicious ones to capture realistic network behavior and apply more tailored strategies, ensuring that benign delays are not mistaken for adversarial actions. Malicious nodes intentionally disrupt consensus by sending false information, whereas benign straggler nodes may delay or fail to send responses due to network latency, resource constraints, or temporary failures without malicious intent. This distinction allows our model to differentiate between non-malicious performance issues and genuine attacks, targeting security measures appropriately.

Building on the threat model of stragglers and malicious nodes described above, we assume that *up to* 40% of nodes in the network are configured as potential stragglers to represent worst-case performance scenarios. However, statistically, not all of these nodes experience delays simultaneously, resulting in an effective number of stragglers that remains well below 40%. Combined with scenarios involving up to 30% malicious nodes, the aggregate proportion of non-responsive or adversarial nodes is highly likely to stay below 50%, preserving a majority of honest and responsive nodes.

This ensures that our majority voting mechanism and consensus process remain reliable and secure, even under adverse conditions.

The simulations are run on a Java-based simulator, JABS [69], a modular blockchain network simulator with the capability of handling around 10000 nodes. For our optimization in Stage 1, we use GAMS [70] and solve the problem of transaction selection on its Java API, which has built-in support for functions in our problem, including regularized gamma functions and standard normal cdf. We assume that there is enough bandwidth for sending and receiving the data required for consensus, and the latency is negligible compared to the block size. Also, to choose the system parameters we used the Ethereum live data set [71] and [72], [73], and set our simulation parameters as follows.

At the beginning of each epoch, the BS collects  $n = 500$  transactions to solve the problem in (2.9) to select a subset of them for further processing by miners. This batch of  $n = 500$  transactions are either all newly generated transactions or a combination of new and backlogged transactions. Denote the number of backlogged transactions at the beginning of epoch  $t$  by  $Y(t)$ . The number of the newly created transactions at the beginning of epoch  $t$  is then  $500 - Y(t)$ . For each transaction  $j$  we generate six numbers representing its fee  $f_j$ , age  $a_j$ , vitality  $v_j$ , amount of computation resource required for processing  $\xi_j$ , size  $\eta_j$ , and maximum depth  $d_j$  following the distributions described below. In case a transaction is backlogged in an epoch, it will preserve these attributes in future epochs until it is selected for processing by the BS.

We assume that a transaction's fee and age are related to its vitality as follows:  $f_j \sim \text{Exp}(20/v_j)$ , truncated between 0 to 100;  $a_j \sim \text{Exp}(6/v_j)$ , truncated between 1 to 32. For equations (2.7) and (2.8), the vitality  $v_j$  of transaction  $j$  is uniformly distributed in  $\{1, 2, \dots, 10\}$  with 10 being the highest vitality. Also,  $\xi_j \sim \text{Gamma}(1.5, 42000)$ ,  $\eta_j \sim \mathcal{N}(3000, 1000)$ . Denote the last confirmed block before the start of epoch  $t$  by  $\mathcal{W}$ . Then at the beginning of epoch  $t$ ,  $500 - Y(t)$  new transactions must be generated. We divide the  $500 - Y(t)$  transactions into five groups of equal size with different maximum depths. For transaction  $j$  in group  $\kappa$ ,  $d_j \sim \text{Poisson}(\mathcal{W}(0.95 - 0.23\kappa))$  for  $\kappa = 0, 1, \dots, 4$ . Moreover, in problem (2.9) the values of  $C = 6.7 \times 10^6$ ,  $S = 1.2 \times 10^6$  are fixed for all the epochs.

However, the value of  $D$  changes in each epoch according to the number of backlogged transactions. That is, if the majority of the batch of 500 transactions for selection by the BS in problem (2.9) are from the backlogged transactions, we continue to increase  $D$  and set it to its next value from the set of values  $\{t - W_\ell, t - W_{(\ell-1)}, \dots\}$  until the majority of the batch of 500 transactions are new transactions.

Furthermore, in equation (2.11), the initial value of  $p_j$  for miner  $j$  when it first joins the system is uniform between 0.5 and 1,  $\beta = 0.1$ , and  $\epsilon = 0.01$  in equation (2.13).

Finally, for the proposed scheme, the simulated networks in Sections V-B and V-C are dynamic, and Section V-C considers a non-dynamic network.

### 2.5.2 Storage

In IoT blockchains, miners often overlap with users submitting transactions. Due to the diverse capabilities of IoT users, some may struggle with storage as the blockchain grows. General solutions to blockchain storage issues typically involve pruning [74] (ignoring old block data) or using side-chains [75], both of which introduce drawbacks [76]. Here, we compare our scheme to Polyshard and LCB regarding storage usage fraction  $R_s$ , defined as the storage space needed per node in a coded blockchain relative to an uncoded conventional blockchain.

In LCB, all miners store identical coded blocks, so coding does not reduce storage. Hence, the storage usage fraction is  $R_s = 1$  for LCB. In sharded blockchains, the number of shards depends on the network size and the fraction of malicious miners. In Polyshard, storage constraints are also influenced by the verification function's polynomial degree. For Polyshard,  $R_s = \frac{1}{\lfloor \frac{(1-2\mu)N(t)-1}{U} \rfloor}$  [12], where  $\mu$  is the fraction of dishonest miners and  $U$  is the maximum degree of the verification function. The storage usage fraction of our scheme is given by:

$$R_s = \frac{Q - \sum_{\ell \in \mathcal{W}} W_\ell + |\mathcal{W}|}{Q}, \quad (2.17)$$

where  $Q$  is the last confirmed block,  $\mathcal{W}$  is the set of encoded blocks, and  $W_\ell$  is the number of

consecutive blocks in the  $\ell$ -th group of encoded blocks.

Fig. 4 shows the evolution of the storage usage fraction  $R_s$  over time in our scheme compared to Polyshard and LCB. We simulated a dynamic blockchain where the numbers of nodes joining and leaving each epoch follow Poisson distributions with  $\lambda_l = 4$  and  $\lambda_e = 10$ , respectively. Starting with 1000 miners, we calculated  $R_s$  across 250 epochs.

Initially,  $R_s = 1$  because blocks are not encoded until the first encoding instant. After encoding starts, miners need significantly less storage to store coded blocks, leading to a steep decline in  $R_s$ . Fluctuations in  $R_s$  occur because miners temporarily store all blocks in a group until the BS encodes them. Once encoded, miners erase these blocks and keep only a coded block, reducing  $R_s$ . This fluctuation repeats with every group.

For Polyshard,  $R_s$  for the balance-checking function [12] is shown with  $U = 1$ , and for  $\mu = 30\%$  and  $\mu = 40\%$  (fractions of dishonest miners). In Polyshard,  $R_s$  decreases as the fraction of dishonest nodes increases, while in our scheme,  $R_s$  remains unaffected by the number of dishonest miners. Thus, with few dishonest miners, Polyshard and our scheme show comparable  $R_s$ . However, as the fraction of dishonest miners grows, our scheme achieves greater storage savings.

Additionally, Polyshard's storage usage increases with higher degrees of the verification polynomial (larger  $U$ ), and the comparison here is based on Polyshard's optimal case,  $U = 1$ .

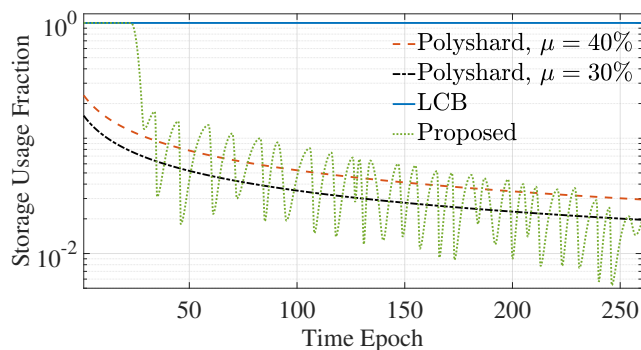


Figure 2.4: Storage usage fraction comparison with  $N = 1000$ ,  $\lambda_l = 4$ , and  $\lambda_e = 10$ .

### 2.5.3 Decentralization

We differentiate between decentralized and distributed systems. Decentralized systems involve decision-making across multiple nodes, with each node determining its behavior and collectively influencing the system, ensuring no single node holds absolute decision-making power. Distributed systems, however, refer to processing spread across nodes, but decision-making may still be centralized or decentralized.

In both Polyshard and LCB, computation is distributed but not truly decentralized. For Polyshard, block generation is unclear, and computations are only distributed during verification. If a conventional consensus algorithm (e.g., PoW, PoS, PBFT) is used for block generation, Polyshard becomes centralized due to the inherent centralization issues of these algorithms [77].

In LCB, a well-equipped node can undertake multiple mining tasks, increasing its contribution to block generation or verification and pushing the network toward centralization. We compare decentralization in LCB with our scheme, while for Polyshard, analysis is limited to descriptive observations due to its ambiguous block generation process.

Denote  $\phi_j(t)$  as the number of blocks generated by miner  $j$  at epoch  $t$ . We adopt the following two metrics to measure decentralization [55]. The Gini coefficient at epoch  $t$  is defined as:

$$I(t) = \frac{\sum_{i=1}^{N(t)} \sum_{j=1}^{N(t)} |\phi_i(t) - \phi_j(t)|}{2N(t) \sum_{j=1}^{N(t)} \phi_j(t)}. \quad (2.18)$$

To provide intuitive understanding, the Gini coefficient is a statistical measure originally used to assess income inequality. In our context, it evaluates how evenly block generation is distributed among miners. A *lower Gini coefficient* signifies that blocks are generated more uniformly across miners, reflecting higher decentralization since no single miner or small group dominates the process. Conversely, a *Gini coefficient close to 1* indicates that block generation is concentrated among a few miners, pointing to centralization.

On the other hand, the entropy at epoch  $t$  is defined as:

$$E(t) = - \sum_{j=1}^{N(t)} \frac{\phi_j(t)}{\sum_{j=1}^{N(t)} \phi_j(t)} \log_2 \frac{\phi_j(t)}{\sum_{j=1}^{N(t)} \phi_j(t)}. \quad (2.19)$$

To further elucidate, *entropy* is a concept from information theory that measures the randomness or unpredictability within a system. In our scenario, higher entropy indicates a more random and evenly spread distribution of mining power among miners. This randomness ensures that block generation is less predictable and more uniformly distributed, signifying greater decentralization. *Higher entropy* thus reflects that no single miner holds a significant advantage, whereas *lower entropy* suggests predictability and potential centralization.

We simulated the LCB and our proposed scheme for 500-time epochs with  $N = 500$  miners in a fast dynamic network with  $\lambda_l = 10$  and  $\lambda_e = 20$ . For our scheme, denote by  $\mathcal{K}_v(t)$  the set of transactions out of  $K(t)$  selected transaction by the BS that is verified by the network and appended to the chain as one block which corresponds to the 1's in  $v^*(t)$ . Therefore, for any  $x_i \in \mathcal{K}_v(t)$ , we give participation credit to miner  $j \in \mathcal{M}_i$ , if miner  $j$  voted in favor of validity of  $x_i$ . Finally, the sum of all credits that miner  $j$  receives from all  $x_i \in \mathcal{K}_v(t)$  form  $\phi_j(t)$ . For the LCB scheme, assume that the time that miner  $j$  takes to finish the assigned tasks at epoch  $t$  follows an exponential distribution with mean  $h_j(t)$ . Denoting the number of total tasks at epoch  $t$  by  $H(t)$ , then the number of assigned tasks to miner  $j$  for epoch  $t$  is  $a_j(t) = \frac{1/h_j(t)H(t)}{\sum_{j=1}^{N(t)} 1/h_j(t)}$ . Hence miners with smaller delays are assigned more mining tasks. Then following [20], we set a deadline to finish the assigned tasks, and all returned tasks after the deadline will be discarded. We set  $\phi_j(t)$  as the percentage of successfully finished assigned tasks by miner  $j$  from all finished tasks by all miners, indicating its participation in decision-making and decentralization. Adopting the parameters in the simulation section of [20],  $H(t) = 10 + 3.2N(t)$ , the deadline for returning the assigned task is 5,  $h_j(t) \sim \mathcal{N}(4.5, 1.66)$  and  $h_j(t) \sim \mathcal{N}(2.5, 5/8)$  for straggler and non-straggler miners, respectively. Also, in rare cases, if  $h_j(t) < 0$ , we assume that the assigned task for miner  $j$  is returned on time. Fig. 5 (a) shows the histograms for the Gini coefficients for the two schemes. It is seen that our scheme has a much smaller Gini coefficient than LCB. In fact, the Gini coefficient of our scheme is close to zero, indicating excellent decentralization. Moreover, in Fig. 5 (b), the histograms of the entropies of the

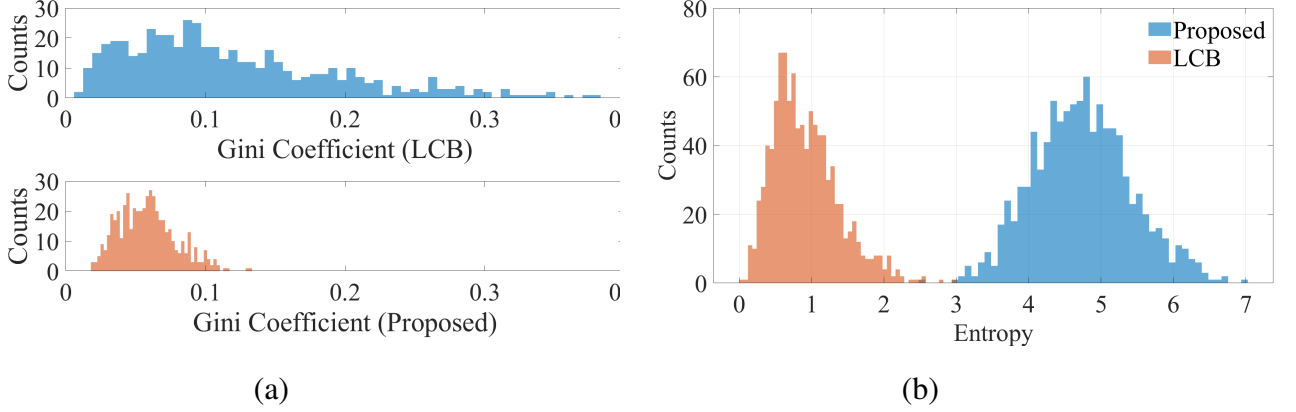


Figure 2.5: The comparison of decentralization between LCB and the proposed scheme with  $N = 500$ ,  $\lambda_l = 10$ ,  $\lambda_e = 20$  and for 500-time epochs, a) Gini coefficient, and b) entropy.

two schemes are shown. It is seen that our scheme has a larger mean and larger variance than LCB, indicating a better decentralization.

#### 2.5.4 Throughput

Throughput is defined as the average number of transactions correctly confirmed per epoch. In uncoded blockchains, throughput primarily depends on the runtime of consensus algorithms. For example, in PoW blockchains, the time required to find the target nonce for solving the hash puzzle determines transaction confirmation time. In coded blockchains, however, coding and decoding complexities significantly influence confirmation time and throughput. Next, we explain how the throughput of the three schemes was obtained through simulations.

For all three schemes, the network has a dishonest miner percentage  $\mu = 30\%$ , with up to 40% straggler miners as described in Section V-A. Simulations involve a ledger size of 10000 epochs.

In Polyshard, the verification function has degree  $U = 1$  (balance-checking). The number of shards is  $S = \lfloor (1 - 2\mu)N - 1 \rfloor \in [15, 165]$ , based on the miner count  $N$ . Each shard proposes a block with  $\frac{500}{S}$  transactions. Following the Polyshard scheme, miners generate and broadcast intermediate results. These results address stragglers (erasure) and dishonest miners (error) via Reed-Solomon decoding (dimension  $S$ , length  $N$ ). A block is valid if the network collectively confirms its validity for each shard  $i \in [1, S]$ . Let  $\varrho(N)$  represent the total confirmed transactions

across all valid blocks for  $N$  miners. Polyshard's throughput is calculated as  $\frac{\bar{\rho}(N)}{500}$ , where  $\bar{\rho}(N)$  is averaged over 1000 simulation rounds.

In LCB, block generators and validators are equal in number. Following the setup in [20], the block generation/validation polynomial degree is 2. The number of block generation and verification tasks is  $N + 5$  (based on equations (5) and (6) in [20]), meeting the security and resiliency threshold of LCC. Tasks are assigned as per Section V-C, using the same straggler distributions. Each miner decodes coded blocks and verifier results to confirm blocks. For block  $B(t)$ , verifier  $j$ 's vote  $\Psi_j(t) \in \{0, 1\}$  determines confirmation via the consensus rule in [20]:  $B(t)$  is valid if decoded successfully and  $\sum_{j=1}^N \frac{a_j(t)}{H(t)} \Psi_j(t) \geq 0.5$ , where  $a_j(t)$  and  $H(t)$  are defined in Section V-C. LCB throughput is the average confirmed transactions per block over 1000 rounds, divided by 500.

In our scheme, throughput is based on correctly confirmed transactions  $\mathcal{K}_v(t)$  from a batch of  $n = 500$  transactions, simulated with and without transaction selection. Without transaction selection, the optimization in Section IV-A is skipped, and transactions are distributed using equation (2.13). With selection, the BS solves problem (2.9) to choose a subset of transactions. Similar to Polyshard and LCB, throughput is averaged over 1000 rounds of simulations.

In Fig. 6, we show the throughputs for the three schemes as a function of the number of miners  $N$ . The throughput of the proposed scheme is higher than Polyshard and LCB. A couple of reasons led our scheme to have higher throughput than the other schemes. One is that we run the blockchain transaction-based, so the validity status of one transaction does not impact other transactions. While in the two other schemes having just one invalid transaction in one block causes the whole block to be invalid, thus making all effort and time spent in vain. Another reason is that in our scheme, we use raptor codes with a linear time encoding and decoding complexity, while the other two schemes employ LCC, which has a higher computational complexity than raptor codes. For instance, the complexity of decoding a length- $N$  Reed-Solomon code at each node to find the collective decision regarding the validity of  $S$  proposed is  $\mathcal{O}(N \log^2 N \log \log N)$  [12]. However, the complexity of finding  $W_\ell$  raw blocks in one group, which are coded by raptor code, is  $\mathcal{O}(\frac{N}{(1+\varepsilon)} \log(1/\varepsilon))$ , where  $\varepsilon$  is the threshold for probability of failure [58]. Lower complexity helps the computation for each

miner take less time, leading to a higher throughput.

Moreover, it seems that the transaction selection in stage 1 of the proposed scheme can significantly increase the throughput since results in faster and less futile work. Also, though when the number of miners is small, e.g.,  $N = 50$ , the proposed scheme without transactions selection has a lower throughput than LCB and Polyshard, for networks with a large number of miners, e.g.,  $N > 400$ , it outperforms LCB and Polyshard. The reason is that the slope of the throughput curve for the proposed scheme without transactions selection is higher than that of LCB and Polyshard.

Finally, it should be noted that the increase in throughput as the number of miners grows reflects the inherent scalability of our sharding-inspired design, which assigns subsets of transactions to different miners and processes them in parallel. As the network expands, more miners can operate concurrently on distinct transaction subsets, effectively reducing bottlenecks and speeding up transaction validation and block creation. This parallel transaction processing is analogous to sharded blockchain systems, where increasing shards or processing units leads to higher aggregate throughput. Our scheme further optimizes this process through coded computation, dynamic miner assignment, and prioritized transaction selection, which together mitigate coordination overhead and computational latency. Consequently, despite the intuition that a larger network might complicate coordination, the system leverages additional resources to enhance overall performance, resulting in the observed increase in throughput with a growing number of miners.

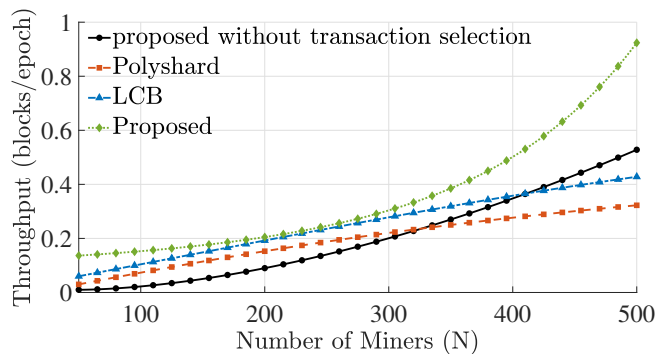


Figure 2.6: Throughput versus the number of miners  $N$  with the percentage of dishonest miners  $\mu = 30\%$ .

Fig. 7 illustrates how the percentage of dishonest miners,  $\mu$ , impacts throughput in a non-dynamic

network. We measured throughput for a fixed  $N = 500$  miners with  $\mu$  ranging from 5% to nearly 50%, each over 1000 simulation rounds.

Our proposed scheme maintains steady throughput as  $\mu$  increases from 5% to 30% and then declines more rapidly from 30% to 50%, yet it remains significantly higher than the other two schemes. For LCB, we simulated two scenarios based on the distributions in Section V-C by sorting  $h_j(t)$  for all miners. In the first scenario, 80% of honest miners are in the first half of the list and 20% in the second, meaning more powerful miners are mostly honest. In the second scenario, 20% of honest miners are in the first half and 80% in the second, making most powerful miners dishonest. This demonstrates the effect of computational power distribution in LCB. Results show that in the first scenario, LCB maintains similar throughput despite more malicious miners, while in the second scenario, its throughput drops below Polyshard when  $\mu > 40\%$ .

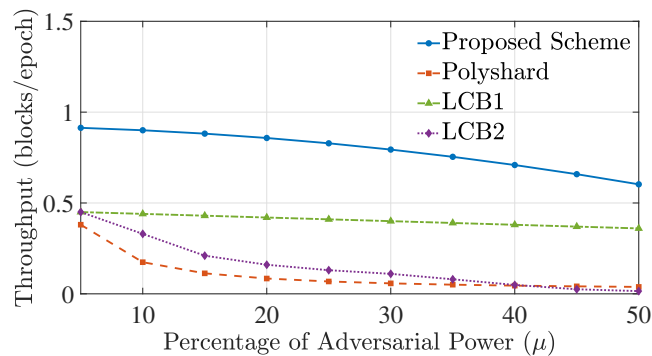


Figure 2.7: Throughput versus the percentage of dishonest miners  $\mu$  with  $N = 500$  miners.

### 2.5.5 Security

Blockchain security has many aspects but can be broadly categorized as malicious and non-malicious behaviors. Malicious behavior involves miners deliberately disrupting the network or appending incorrect blocks. Non-malicious behavior leads to similar outcomes but occurs unintentionally, due to failures or miner straggler effects.

## Discrepancy Attack

The discrepancy attack [43] occurs during block generation. Polyshard assumes all nodes receive the same blocks at the start of each epoch, with blocks proposed for each shard by nodes within that shard. Sharding, which divides the blockchain into parallel sub-chains by splitting miners into smaller groups, makes the network vulnerable to attacks since smaller groups are easier to control. An adaptive adversary can target a shard and generate multiple conflicting blocks, causing network discrepancies. Detecting such discrepancies [43] involves significant communication costs. Preventing this requires the network’s miners to scale as  $S^{\frac{U}{2}}$ , where  $S$  is the shard count and  $U$  the polynomial verification degree—a challenge for real-world blockchains.

The LCB counters this attack using coding in block generation. All miners propose a single block per epoch, formed using the LCC. Thanks to the error resilience of LCC, the network can reliably identify the correct block despite discrepancies.

In our proposed scheme, discrepancy attacks can occur at two levels: transaction and block. At the transaction level, attackers send conflicting votes on a subset of transactions to different miners. At the block level, they send conflicting blocks.

To address transaction-level attacks, our scheme selects the number of miners  $M(t)$  to validate each transaction based on the miners’ estimated reliability during Stage 1. This ensures that even with conflicting votes, the scheme can reach a correct collective decision.

For block-level attacks, all miners broadcast the confirmed block, and the block agreed upon by the majority is appended to the chain. A successful block-level attack would require attackers to control more than half of the network miners, which is unlikely in real-world blockchains.

## Straggler Effect

In both LCB and Polyshard, consensus involves sub-tasks such as computing intermediate results of Lagrangian polynomials at specific points. Polyshard assumes all miners return their results on time. While coding techniques can handle some erasures by treating straggler miners as such, an increase in stragglers reduces the network’s ability to handle erroneous results from adversarial

miners.

LCB addresses this by assigning multiple computation tasks to each miner. Miners with greater computing power handle more tasks, compensating for delays caused by stragglers who fail to deliver results on time.

Our scheme mitigates straggler effects by adjusting the number of miners assigned to validate each transaction based on Equation (2.13). It accounts for stragglers or other network conditions that delay miners by continuously updating their reliability. As the number of stragglers increases, the scheme assigns more miners to process each transaction, ensuring the network can still make accurate decisions.

### **Eclipse and 51% Attacks**

An eclipse attack involves an attacker creating a controlled environment around one or more miners, manipulating them into wrongful actions. A 51% attack occurs when over half of the miners are controlled by a single group, allowing them to dominate decision-making in the network.

Both Polyshard and LCB use LCC, but they differ in handling encoded outputs. Polyshard waits for all encoded results before decoding, while LCB allows each miner to decode using a subset of the fastest coded outputs. This approach makes LCB more vulnerable to eclipse and 51% attacks, especially in IoT blockchains where traditional consensus methods like PoW are absent, making it easier and cheaper to produce erroneous results. By assigning multiple tasks to some miners, LCB risks malicious miners targeting subsets to send incorrect outputs. LCB assumes that faster miners are mostly honest, which may not hold in practice, allowing powerful malicious miners to corrupt the chain by taking on a significant number of tasks.

In contrast, Polyshard assumes secure block generation within shards and sufficiently large shard sizes to prevent 51% attacks, making it resistant to these threats under such assumptions.

In our scheme, random transaction assignment at the BS stage prevents attackers from targeting specific miners, as they cannot predict which miners will validate a transaction. Additionally, decisions rely on majority rule at the block verification stage, making a 51% attack successful only

if adversaries control more than half of all miners, which is highly improbable.

### Attacks in Distributed Encoding

Both LCB and Polyshard use distributed encoding, where miners exchange data, apply LCC, and store the resulting coded ledger locally. Miners then process transactions on the coded data and use LCC for consensus. However, as shown in [78], if dishonest miners send conflicting data to honest miners in a distributed encoding network, honest miners can be confused about the final decision, potentially causing the blockchain to fork at each epoch. Preventing this requires every honest miner to contact all others and wait for their results, negating the performance advantages of waiting only for the fastest miners’ outputs in LCB and Polyshard.

Our scheme, however, does not rely on distributed encoding as described in [78]. Instead, computations are performed on raw data, and each miner encodes past block data independently. This approach makes our scheme resistant to the attack described in [78].

Finally, we summarize whether or not each coded scheme is resistant to various attacks in Table 2.3.

Attack Type	Polyshard	LCB	Proposed
Discrepancy Attack	No	Yes	Yes
Straggler Effect	No	Yes	Yes
Eclipse and 51% Attacks	Yes	No	Yes
Attacks in Distributed Encoding	No	No	Yes

Table 2.3: Resistance to various attacks by different coded blockchain schemes.

### Simulation Comparison

We conducted two experiments to evaluate the impact of attacks on the three schemes. The first experiment examined the effect of a discrepancy attack by measuring throughput as a function of the number of miners, similar to Section V-D, but without straggler miners. All miners were either honest or dishonest, with 20% ( $\mu = 20\%$ ) being dishonest and sending discrepant values to disrupt the network.

We defined  $\vartheta$  as the number of distinct messages a dishonest miner can send during the block generation stage [43]. For Polyshard and LCB,  $\vartheta$  represents the number of different coded blocks a dishonest miner can send. Unlike simulations where dishonest miners send just one invalid block ( $\vartheta = 1$ ), they sent multiple invalid blocks ( $\vartheta > 1$ ) to corrupt consensus and hinder block decoding.

Our scheme, however, differs fundamentally. Transactions are processed directly, with blocks created only after valid transactions are identified. Consequently,  $\vartheta$  is irrelevant in our scheme. Votes on transaction validity are binary: dishonest miners can either incorrectly vote valid transactions as invalid or invalid ones as valid, but they cannot send multiple discrepant messages. This binary voting mechanism eliminates the ability to exploit  $\vartheta$  for a discrepancy attack in our scheme.

We computed the throughput of LCB, Polyshard (for  $\vartheta = \{2, 3\}$ ), and the proposed scheme, with results shown in Fig. 8. Polyshard is vulnerable to attacks, as an increase in miners leads to more dishonest ones (fixed  $\mu$ ), causing more discrepant values and complicating LCC decoding [43]. This reduces successfully decoded blocks and throughput, worsening as  $\vartheta$  rises. For networks with around 300 miners, Polyshard’s throughput drops to near zero.

LCB performs better, leveraging LCC for both block generation and verification. While the attack slightly decreases throughput due to increased decoding complexity with more miners/messages, the decline is slower than in Polyshard, even as  $\vartheta$  rises.

The proposed scheme is immune to discrepancy attacks by processing transactions on raw data, where dishonest miners can only send false votes for their assigned transactions. Unlike Polyshard and LCB, where invalid transactions in blocks affect others, the proposed scheme processes each transaction separately, maintaining throughput even as miners increase.

In the second experiment, with  $N = 500$  miners and varying straggler percentages, throughput was analyzed (Fig. 9). Polyshard showed the largest throughput decline, as it assumes timely shard proposals and task completion. LCB demonstrated stable throughput despite rising stragglers, effectively addressing the issue. The proposed scheme maintained consistent throughput due to its miner assignment method (Section IV-B).

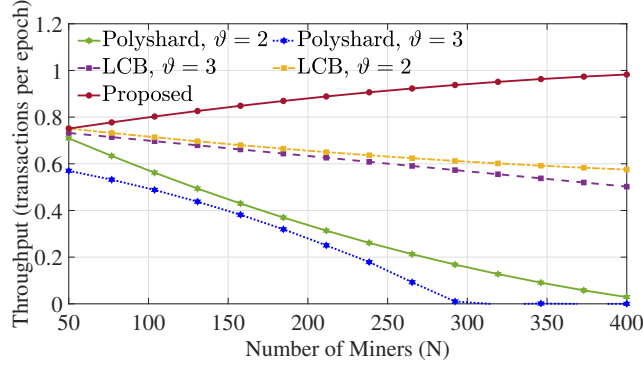


Figure 2.8: The effect of discrepancy attack on the throughput based on the number of the miners ( $N$ ) with  $\mu = 20\%$ .

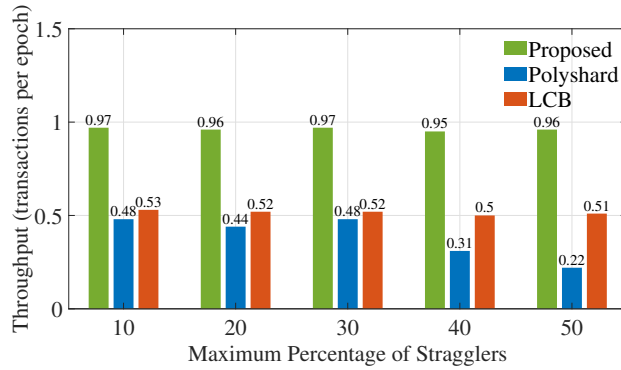


Figure 2.9: The effect of stragglers on the throughput based on the maximum percentage of the straggler miners with  $N = 500$ .

## 2.6 Conclusions

We have proposed a new coded IoT blockchain to simultaneously improve the throughput, security, decentralization, and storage while maintaining the key features of legacy blockchains. The proposed scheme employs the rateless code to encode the block data and store the coded blocks distributedly. Instead of processing the transactions in blocks, as is the case in other existing coded blockchains which may discard the processed blocks when it has a single invalid transaction and hence decreases the throughput, our approach is based on reaching a consensus on each transaction. This fact and an additional transaction assignment stage where a subset of submitted transactions are selected and assigned to miners, as well as low decoding and encoding complexity of raptor codes enable our proposed scheme to achieve higher throughput, distributed storage, and better scalability.

Also, we discussed how performing the verification on raw as opposed to coded data (as in Polyshard and LCB) provides a more secure and decentralized scheme closer to the legacy blockchain.

## **Chapter 3: HybridChain: Fast, Accurate, and Secure Transaction Processing with Distributed Learning**

### **3.1 Introduction**

Distributed ledgers, commonly known as blockchains, have emerged as disruptive technologies capable of revolutionizing various industries. Such decentralized and transparent systems enable secure and immutable recording of transactions without the need for intermediaries such as government agencies. Built on a peer-to-peer network of computers that collectively validate and record transactions, the blockchains employ cryptographic techniques to ensure the integrity and authenticity of the data. Originally developed to create and exchange digital currencies such as Bitcoin [79], blockchains technology now find applications in various domains, making transformative impacts [80].

However, the potential benefits of the blockchain technology are accompanied by challenges, particularly in scalability [81]. Blockchain systems have limited capacity and can process only a limited number of transactions per time unit. This issue is challenging to address due to the blockchain trilemma [54], which refers to the difficulty of achieving high levels of scalability, security, and decentralization simultaneously. Any improvement in one aspect would come at the expense of one or both of the others, and finding the right balance among the three aspects is critical for the success of a blockchain system.

Solutions to addressing these issues can be broadly classified as either off-chain or on-chain [52]. Off-chain solutions improve the scalability by reducing the computational burden on the main blockchain. They include state channels [82], payment channels [83], sidechains [84], and Plasma [85], all of which handle transactions off-chain to reduce congestion on the main chain and thus increase the scalability of the blockchain. On the other hand, on-chain solutions involve changes

to the underlying blockchain protocol, such as changing the block data structure, and employing a new consensus mechanism. In particular, AdaptChain [86], and Jidar [87], compress the size of transaction representation in each block, saving the bandwidth and reducing the propagation delay, thereby increasing the throughput. On the other hand, Algorand [88] and [89] each propose a new consensus algorithm based on Pure Proof of Stake (PPoS) and Votes-as-a-Proof (VaaP), respectively, to increase the throughput.

Moreover, as an on-chain solution, sharding which is adopted in Benzene [90], and Omniledger [5], is the most significant approach among all on-chain and off-chain approaches. Firstly, sharding partitions the network into smaller shards, and each shard operates independently, processing its own subset of transactions and smart contracts. This division of labor reduces the computational burden on individual nodes and increases the overall throughput of the network. Off-chain solutions like state channels and payment channels can help reduce congestion on the main chain but have limited scalability potential. The reason for this limitation is that these solutions depend on direct communication and coordination among the participants, which imposes a cap on the number of nodes that can participate.

Secondly, sharding maintains the security and decentralization benefits inherent in blockchains. Each shard has its own set of validators, ensuring that consensus is reached on the subset of transactions it processes. By distributing the network's computational load across multiple shards, sharding enhances the overall network's resilience against attacks and reduces the risk of a single point of failure. In comparison, off-chain solutions like sidechains and Plasma often introduce additional trust assumptions and require trust in a set of federated entities or a single trusted party, compromising the decentralized nature of the blockchain.

Furthermore, sharding facilitates efficient data storage and retrieval. With the partitioning of the blockchain into smaller shards, each shard only needs to store a fraction of the entire blockchain's data, reducing storage requirements for individual nodes. Additionally, sharding enables parallel processing and faster block propagation within shards, resulting in reduced latency and faster transaction confirmation. Pertaining to this matter [91] delves into the architectural components of

sharded blockchain systems, such as node selection and cross-shard transaction processing, aiming to further enhance scalability and performance. Also, [92] introduces DSSBD, a dynamic state sharding blockchain architecture, utilizing deep reinforcement learning to optimize blockchain efficiency, thereby enhancing throughput and security. However, even though on-chain solutions like AdaptChain, Jidar, and CUB offer optimization techniques to improve throughput, they do not provide the same level of scalability as sharding.

On the other hand, a new avenue gaining significant attention is the Directed Acyclic Graph (DAG) approach. Unlike existing off-chain and on-chain solutions designed for blockchains, DAG is a unique data structure that enables non-linear data recording in a distributed ledger. Transactions in a DAG are interlinked based on their dependencies, forming a directed graph with no cycles. The structure permits parallel processing of transactions, providing a higher throughput than conventional blockchain systems. DAG-based distributed ledgers could address scalability and transaction processing challenges for two reasons. Firstly, as the number of users increases in the network, the transaction processing capacity of a DAG can increase, whereas conventional blockchain systems become less efficient. Secondly, DAGs can offer quicker transaction confirmation compared to blockchains. Transactions in a DAG can be confirmed as soon as they are added to the network without waiting to be processed in a block, as in a blockchain. DAG schemes include Nano [93], Avalanche [94], and IOTA [95].

However, despite the advantages of the sharded blockchains and DAG distributed ledgers, they have a number of drawbacks [96], [97] that will be elaborated in Section II-B. Motivated by these drawbacks, in this chapter, we propose a novel consensus mechanism that provides quick transaction processing like DAG but with improved accuracy and security, achieved by introducing validators as in blockchain. This results in a scalable, efficient system capable of processing high volumes of transactions with high accuracy.

The main contributions of this chapter are as follows:

1. We propose a new transaction validation scheme that prioritizes the transactions more likely to be valid for faster and more efficient processing, thus achieving higher throughput.

2. We propose a lightweight consensus algorithm that utilizes decentralized learning. Validators exchange perceptions to assess transaction conflicts with the witness set, representing input transactions in the UTXO model. These perceptions form an intermediate belief about validity. Validators combine beliefs with each others, leading to local decisions on validity. Final consensus emerges through a majority vote, ensuring accurate and efficient transaction validation.
3. We discuss in-depth the notions of security, decentralization, and throughput and compare our proposed scheme with IOTA and Omniledger in terms of these metrics.

The remainder of the chapter is organized as follows. Section 3.2 provides a brief overview of the conventional blockchain, sharded blockchain, and the DAG distributed ledger. Section 3.3 presents our proposed HybridChain scheme. Section 3.4 compares our scheme with IOTA and Omniledger in terms of security. Section 3.5, compares our scheme with IOTA and Omniledger in terms of scalability of throughput and latency through simulations. Finally, Section 3.6 concludes the chapter.

## **3.2 Backgrounds**

This section first gives an overview of the conventional blockchain, sharded blockchain, and DAG distributed ledger. Then, we discuss the drawbacks of these existing schemes.

### **3.2.1 Existing Approaches**

#### **Conventional Blockchain**

Distributed ledger technologies utilize blockchains to store ordered records called blocks, facilitated by a network of untrusted entities known as validators. These blocks are linked through references to previous block headers, creating an immutable chain. A typical block consists of two main sections: transactions and header. The header contains the hash of the previous block header and the hash of the Merkle root, which represents the digest of all transactions in the block. Validators propose new blocks using a consensus algorithm, selecting transactions and forming the

block header. When creating the next block, validators verify the validity of the proposed block, ensuring it adheres to the underlying protocols. Once confirmed, the validators accept the proposed block as the next block, appending it to the existing chain.

The consensus mechanism used in a blockchain determines how blocks are proposed. There are primarily two types of consensus mechanisms: Nakamoto Consensus and Classical Consensus [98]. Nakamoto Consensus includes Proof of Work (PoW) and Proof of X (PoX), where miners solve mathematical problems or utilize a scarce resource to propose blocks. Classical Consensus, on the other hand, relies on voting and allows for faster confirmation of transactions; Practical Byzantine Fault Tolerance (PBFT) is an example where nodes communicate in quorums to agree on the system's state. Note that Nakamoto Consensus can result in wasted efforts, while classical consensus may suffer from congestion and low throughput. To address these issues, solutions like sharded blockchain and DAG distributed ledgers have been proposed.

Next we provide an overview of Omniledger [5], which is a sharded blockchain, and IOTA [95], which is a distributed ledger based on DAG.

### **Sharded blockchain—Omniledger**

In a conventional blockchain, all validators validate and store every transaction. On the other hand, sharding partitions the blockchain into smaller subsets called shards, where each shard is responsible for processing and storing a subset of transactions. By distributing the workload across validators, sharding enables parallel processing and increases the throughput. Unlike non-sharded blockchains, where every validator processes all transactions, sharding allows validators within a shard to focus solely on the transactions in their shard, resulting in faster confirmation times. Additionally, sharding facilitates scalability by adding more shards as the network expands. However, sharding also introduces new challenges such as shard coordination, secure cross-shard communication, and specialized consensus protocols.

As a sharded blockchain, Omniledger operates in epochs and each epoch consists of four main stages. Initially, validators are assigned to shards, and leaders are selected using the *RandHound*

distributed randomness beacon. In the second stage, these leaders propose new blocks containing transactions limited to their respective shards. Validation of the proposed block within each shard is conducted by validators, employing the *ByzCoinX* consensus algorithm, followed by voting on its validity. Once a two-thirds majority of validators agree, the block is added to the blockchain. The third stage addresses cross-shard transactions through the utilization of *Atomix*, ensuring atomicity across multiple shards. In this stage, the initiating shard secures the relevant accounts, creates an *Atomix* instance, and requests participation from other shards involved in the transaction. Execution of the transaction occurs when all participating shards reach an agreement, and upon success, the *Atomix* instance is committed. In the final stage, the epoch is concluded, resulting in the update of the blockchain state and preparation for the next epoch.

## **DAG—IOTA**

In IOTA, the constructed Directed Acyclic Graph (DAG) is referred to as the Tangle, where vertices represent transactions. In the Tangle, a directed edge from vertex  $v_i$  to  $v_j$  signifies that transaction  $v_i$  validates transaction  $v_j$  [95].

In IOTA, the notion of validators associated with traditional blockchains does not exist. Instead, the ledger relies on a network of nodes. i.e., devices or computers, to submit and process transactions. When nodes submit transactions, they are verified by other nodes within the network. This self-validating mechanism ensures the legitimacy and consistency of transactions. IOTA consists of two types of nodes: light nodes and full nodes. Light nodes are responsible for expanding the Tangle by generating and attaching new transactions to it, while full nodes store all the data in the Tangle. Light nodes rely on full nodes to retrieve the necessary information.

Similar to the concept of confirmed blocks that are irrevocable in blockchains, nodes in IOTA must reach consensus on the set of confirmed transactions. To achieve this consensus, for each generated transaction in IOTA, light nodes fetch the required information from full nodes and solve a Proof of Work (PoW) puzzle. They then select two older transactions, called parents, from the tip transaction pool[14]. Here, a tip transaction refers to a transaction that has been broadcast to

the network but has not yet been validated by being directly or indirectly referenced by another transaction. This parent selection in the Tangle allows for the existence of two sets of transactions associated with a given transaction: the past cone, which consists of transactions referenced directly or indirectly, and the future cone, which includes transactions that reference the given transaction either directly or indirectly.

To finalize the consensus, IOTA employs a trusted node, known as the Coordinator, that periodically issues special transactions called milestones, which act as checkpoints for the Tangle. Transactions that are in the past cone of these milestones are considered confirmed.

It is worth noting that since nodes must solve a PoW puzzle to attach the transactions to the Tangle, the consensus mechanism in IOTA is classified as a Nakamoto consensus.

### 3.2.2 Drawbacks of Existing Schemes and Motivation of Proposed Approach

While DAG schemes are capable of decreasing latency by removing validators and processing transactions solely based on their positions in the submission order graph, this results in reduced throughput, especially when there are dishonest nodes that behave strategically. On the other hand, a sharded blockchain, in which validators process transactions in parallel, can improve throughput scalability while keeping latency low in such circumstances.

However, the processing of cross-shard transactions poses a significant challenge for sharded blockchains. Specifically, the shards must exchange information about the transaction's validity and the associated account balances to process cross-shard transactions. This necessitates a mechanism for cross-shard communication and consensus, which can be intricate and time-consuming. Moreover, a large volume of cross-shard transactions can potentially create a bottleneck in the system, as the shards must collaborate and communicate effectively to ensure that all transactions are processed accurately and promptly. Omniledger aims to address the challenges associated with cross-shard transactions by implementing Atomix processing. However, as we will demonstrate in Section V, cross-shard transactions in Omniledger can still introduce significant network delays, particularly when the number of shards increases and a larger volume of transactions are submitted as cross-shard.

Moreover, the notion of decentralization in a distributed ledger covers both storage and computation. Decentralizing storage involves distributing data across multiple nodes with minimal redundancy so that each node stores only a fraction of the ledger data. Decentralizing computation involves distributing the computing process for transactions among validators in the network.

Omniledger exhibits good decentralization of both storage and computation since, as a sharded blockchain, validators process transactions for their respective shards and store the associated data. On the other hand, for IOTA DAG where validators are not present to process transactions, edges of the DAG graph created by nodes in the network validate or reject vertices corresponding to transactions. Decentralization of computations in IOTA can be achieved to some extent by ensuring that the decision to accept or reject a vertex is not made by a single or small group of nodes.

Furthermore, IOTA lacks adequate storage decentralization, since light nodes in IOTA heavily depend on full nodes to send or receive required information for selecting transaction parents and updating their perception of the Tangle. Additionally, as explained in Section II-A-2, the decentralization of computations within IOTA is governed by the Coordinator, a centralized entity responsible for determining the set of confirmed transactions. Consequently, it becomes evident that IOTA lacks computational decentralization.

Additionally, as explained in Section IV, Omniledger is found to be vulnerable to replay attacks and message withholding attacks. On the other hand, IOTA faces its own set of vulnerabilities, namely orphanage attacks and routing attacks.

Our proposed HybridChain in this chapter aims to tackle the aforementioned challenges by adopting a hybrid approach that combines the benefits of both DAG and sharded blockchains. Similar to DAG-based schemes, our scheme has the advantage of low latency and quick transaction processing. Additionally, our scheme features parallel transaction processing by validators, akin to sharded blockchains, resulting in decentralized storage and computations. We also ensure sufficient redundancy in the data storage and employ a consensus mechanism that enables our scheme to withstand attacks that both DAG and sharded blockchains are susceptible to. Table 3.1 summarizes the level of scalability, decentralization, and security of the three schemes.

Scheme	Trilemma				
	Scalability		Decentralization		Security
	Throughput	Latency	Storage	Computation	
IOTA	Moderate	Good	Poor	Moderate	Poor
Omniledger	Moderate	Poor	Good	Good	Poor
HybridChain	Good	Good	Good	Good	Good

Table 3.1: Comparison of IOTA, Omniledger, and the proposed HybridChain based on the trilemma of scalability, decentralization, and security.

### 3.3 HybridChain

This section describes our proposed HybridChain in detail. We first provide an overview and then describe the transaction attributes that are used by HybridChain. Next, we elaborate on the transaction processing procedure and how the system parameters are updated. Finally, we describe how consensus is reached by validators.

#### 3.3.1 Overview

Suppose that the network consists of  $N$  users  $U = \{u_1, \dots, u_N\}$  that submit transactions, and  $M$  validators  $\mathcal{M} = \{m_1, \dots, m_M\}$  that process transactions. Note that these two sets can overlap. Corresponding to each validator or user we define a reliability value  $\rho_i^m \in [0, 1], i = 1, \dots, M$  or  $\rho_j^u \in [0, 1], j = 1, \dots, N$ . Each validator can exchange information with all other validators in the network. Each validator is either honest or dishonest: honest validators strictly adhere to all network rules and protocols, while Byzantine dishonest validators act in a random and coordinated fashion. The Byzantine validators possess comprehensive knowledge of the network, including the network structure, the algorithms employed, and the exchanged data. We assume that at most  $f \leq \lfloor M/2 \rfloor - 1$  validators are Byzantine dishonest.

Validators use the UTXO model to store past transactions in the ledger, where each transaction  $x_i$  is created by some of the past transactions  $\{x_1, x_2, \dots, x_{i-1}\}$ . Denote  $W_i \subset \{x_1, \dots, x_{i-1}\}$  as the witness set of  $x_i$ . That is,  $x_i$  is valid if it is not in conflict with any of the transactions in  $W_i$ .

Similar to the sharded blockchain, we partition time into epochs, but with variable durations.

Each epoch comprises several rounds, and the number of rounds within each epoch can vary. In each epoch validators process  $\lambda = \lfloor M/(2f + 2) \rfloor$  transactions (the choice of this value is explained in Remark 5 in Section III-D). Denote the batch of transactions that is processed during epoch  $n$  by  $X(n) = \{x_{(n-1)\lambda+1}, x_{(n-1)\lambda+2}, \dots, x_{n\lambda}\}$ . Then for this epoch, validators are randomly grouped into  $\lambda$  communities of size  $M/\lambda$  and validators in one community process a single transaction in  $x_\ell \in X(n)$ . Denote the set of validators in the community that is responsible for processing transaction  $x_\ell \in X(n)$  by  $\mathcal{V}_\ell$ . Then validators in  $\mathcal{V}_\ell$  process  $x_\ell$  in maximum  $|W_\ell|$  rounds, and if  $x_\ell$  is validated, then it is stored by all validators in  $\mathcal{V}_\ell$ . Since all  $\lambda$  transactions in  $X(n)$  are processed in parallel across communities, the maximum number of rounds in epoch  $n$  is  $R_n = \max\{|W_{(n-1)\lambda+1}|, |W_{(n-1)\lambda+2}|, \dots, |W_{n\lambda}|\}$ .

In each round  $r = 1, \dots, R_n$ , validator  $m_k \in \mathcal{V}_\ell$  computes a quantity  $p_k^\ell(r) \in [0, 1]$  that represents the current actual belief of validator  $m_k$  about the validity of transaction  $x_\ell$ . (The calculation of  $p_k^\ell(r)$  is given in Section III-C.) Then the local decision by validator  $m_k$  at round  $r$  about the validity of transaction  $x_\ell$ , denoted by  $v_k^\ell(r)$ , is given by:

$$v_k^\ell(r) = \begin{cases} 1, & \text{if } p_k^\ell(r) \geq \eta_1(\mathbf{a}_\ell, \mathbf{y}_k) = \mu_1 - \frac{\sigma(\mathbf{a}_\ell^T \mathbf{y}_k)}{2}, \\ 0, & \text{if } p_k^\ell(r) \leq \eta_2(\mathbf{a}_\ell, \mathbf{y}_k) = \mu_2 - \frac{\sigma(\mathbf{a}_\ell^T \mathbf{y}_k)}{2}, \\ \phi, & \text{otherwise,} \end{cases} \quad (3.1)$$

where  $\phi$  denotes that a local decision cannot be made,  $\eta_1(\mathbf{a}_\ell, \mathbf{y}_k)$  and  $\eta_2(\mathbf{a}_\ell, \mathbf{y}_k)$  are the acceptance and rejection thresholds, respectively, with  $\mu_1$  and  $\mu_2$  being some fixed values,  $\sigma(\cdot)$  being a sigmoid function,  $\mathbf{a}_\ell$  being an attribute vector for transaction  $x_\ell$  and  $\mathbf{y}_k$  being a weight vector associated with validator  $m_k$ .

The final collective decision, i.e., the network consensus about the validity of  $x_\ell$ , can be reached

at round  $r$  if either of the following two conditions is met:

$$V^\ell = \begin{cases} 1, & \text{if } \sum_{m_k \in \mathcal{V}_\ell} \mathbf{1}_{[v_k^\ell(r)=1]} > \frac{1}{2}|\mathcal{V}_\ell|, \\ 0, & \text{if } \sum_{m_k \in \mathcal{V}_\ell} \mathbf{1}_{[v_k^\ell(r)=0]} > \frac{1}{2}|\mathcal{V}_\ell|, \end{cases} \quad (3.2)$$

where  $V^\ell = 1$  means that transaction  $x_\ell$  is confirmed by the network and  $V^\ell = 0$  otherwise, and  $\mathbf{1}_{[\cdot]}$  is the indicator function.

In Fig. 3.1, we illustrate the transaction processing mechanism of HybridChain at epoch  $n$ . Each epoch encompasses  $\lambda$  transactions, represented by circles. Transactions validated by the network are marked in green, signifying consensus on their validity, while red circles indicate rejected transactions, illustrating the network's decision not to store these in the nodes due to their invalidity. The figure also highlights transactions in yellow, representing those currently under verification or processing within epoch  $n$ . For instance, transaction  $x_{(n-1)\lambda} + 1$  has a witness set comprising transaction  $x_1$ , indicating its solitary verification source. Conversely, transaction  $x_{(n-1)\lambda} + 2$  is verified by a larger witness set, including transactions  $x_\lambda$ ,  $x_{2\lambda}$ , and  $x_{(n-2)\lambda+2}$ . Directed edges in the figure connect each transaction to its corresponding witnesses, exemplifying a simplified representation of the underlying DAG structure, where a comprehensive view would depict all epochs and transactions.

Communities, shown in the figure as grouped nodes, are assigned to process transactions within each epoch. For epoch  $n$ , each community, consisting of  $M/\lambda$  nodes, is tasked with processing one transaction. While the figure illustrates an orderly node assignment to communities, actual assignments are randomized for efficiency and security. This depiction serves to elucidate HybridChain's dual approach: transactions are individually processed based on conflicts with their respective witness sets (akin to parent nodes in a DAG), while simultaneously enabling parallel and sharded processing across multiple communities.

### 3.3.2 Attribute Vector

Here we discuss the idea behind using the attribute vector  $\mathbf{a}_\ell$  for transaction  $x_\ell$ . Unlike existing distributed ledger schemes that spend the same effort to process all submitted transactions, our scheme adjusts the time and computational resources. The idea is that all submitted transactions do not need the same amount of processing, and some are more likely to be valid than others. Therefore, transactions that are more likely to be valid can be treated less strictly to save time and other resources. In this regard, we use the attribute vector  $\mathbf{a}_\ell$  for transaction  $x_\ell$  to find its threshold of acceptance or rejection. The elements of  $\mathbf{a}_\ell$ , along with their descriptions, are listed in Table 3.2.

Here we explain the intuition behind the relation between each element of  $\mathbf{a}_\ell$  and the validity of transaction  $x_\ell$ . For each element of  $\mathbf{a}_\ell$ , a higher value indicates a higher likelihood of  $x_\ell$  being valid. Consequently, the acceptance and rejection thresholds for transaction  $x_\ell$  in (3.1) will be lower.

$a_\ell[1]$  is the inverse of the value of the transaction  $x_\ell$ . A transaction triggering a larger amount of asset transfer between two users should be processed more carefully. However, while comparing two transactions, especially the ones that have lesser values, the transaction with higher fees can decrease

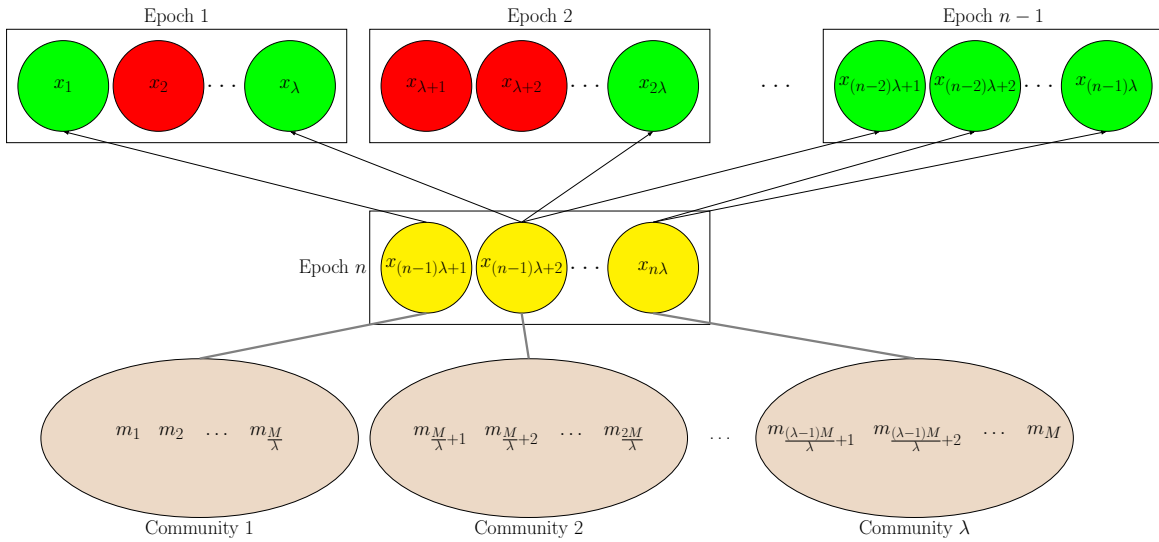


Figure 3.1: Transaction Processing in HybridChain at Epoch  $n$ . Green circles represent validated transactions, red circles denote rejected ones, and yellow circles indicate transactions under verification. Directed edges show witness relationships, simplifying the DAG structure. Brown ovals represent communities responsible for parallel and sharded transaction processing, highlighting HybridChain’s integrated approach combining sharding and DAG for efficient, secure consensus.

Notation	Description
$a_\ell[1]$	The inverse of the value of the asset that is transferred from sender(s) to receiver(s)
$a_\ell[2]$	The amount of the fee that a transaction submitter pays to validators for processing
$a_\ell[3]$	The elapsed time from the previous transaction submitted by the same user in number of rounds
$a_\ell[4]$	The inverse of the witness set size
$a_\ell[5]$	The weighted average of reliability of the user(s) that initiate the transaction

Table 3.2: Elements of the attribute vector  $\mathbf{a}_\ell$ .

the likelihood of invalid transaction. Hence  $a_\ell[2]$  is the fee that must be paid to the validators for processing transaction  $x_\ell$  regardless whether  $x_\ell$  will be confirmed or not.  $a_\ell[3]$  is the elapsed time from the previous transaction submitted by the same user in number of rounds. This means if one user submits many transactions in a short time, it can be a sign of abnormality.  $a_\ell[4]$  is the inverse of the size of the witness set  $W_\ell$  for transaction  $x_\ell$ . Since a transaction with a bigger witness size  $|W_\ell|$  is triggering a bigger number of unspent transactions  $x_j \in W_\ell$ , then such transaction should be handled more prudently. In general, transaction  $x_\ell$  can be initiated by multiple users from the set  $U$ . This means that each transaction in the witness set  $W_\ell$  can have a different owner, which can be considered analogous to a multi-signature transaction in the UTXO model. Specifically, for each  $a_\ell[5]$ , represents the weighted average of the reliability of the sender(s) transaction  $x_j \in W_\ell$ . Let  $\mathcal{U}(x_j)$  denote the index of the user who owns  $x_j$ , and  $\mathcal{A}(x_j)$  represent the value of  $x_j$ . Then

$$a_\ell[5] = \sum_{x_j \in W_\ell} \frac{\mathcal{A}(x_j)}{\sum_{x_j \in W_\ell} \mathcal{A}(x_j)} \rho_{\mathcal{U}(x_j)}^u.$$

### 3.3.3 Transaction Validation Procedure

Recall that in each epoch  $n$ , each transaction  $x_\ell \in X(n)$ , as well as information on its witness set  $W_\ell$  is available to all validators. Therefore, for any  $x_j \in W_\ell$ , if validator  $m_k \in \mathcal{V}_j$ , then it can determine whether or not  $x_\ell$  and  $x_j$  are in conflict, and we define the perception:

$$q_k(x_\ell, x_j) = \begin{cases} 0, & \text{if } x_\ell \text{ and } x_j \text{ are in conflict} \\ 1, & \text{if } x_\ell \text{ and } x_j \text{ are not in conflict;} \end{cases} \quad (3.3)$$

otherwise validator  $m_k$  cannot discern. Moreover, denote by  $W_\ell^k$  a random permutation of the elements in  $W_\ell$  generated by validator  $m_k$ . That is, for two validators  $m_k$  and  $m_{k'}$ ,  $W_\ell^k$  and  $W_\ell^{k'}$  have the same elements but with different orderings. We denote the transaction corresponding to the  $r$ -th element in  $W_\ell^k$  by  $W_\ell^k(r)$ .

### Processing procedure in each round $r$

After each round  $r$  in epoch  $n$ , one or more transactions may be eliminated from the set of the transactions  $X(n)$  since their processing is finished by their corresponding validators, i.e.,  $v_k^\ell(r) = 0$  or 1 in (3.1). We denote  $X_r(n)$  as the set of transactions still being processed after round  $r$  of epoch  $n$  with  $X(n) = X_1(n)$ . Then for each round  $r$  in epoch  $n$ , the processing of all transactions in  $X_r(n)$  consists of the following four stages:

#### Stage 1: Validators in all communities exchange perceptions:

*Step 1: Sending perceptions to other validators:* For each transaction  $x_\ell \in X_r(n)$ , and for all transactions  $x_{\ell'} \in X_1(n)$  each validator  $m_k \in \mathcal{V}_\ell$  checks if  $r \leq |W_{\ell'}^k|$  and  $m_k \in \mathcal{V}_{W_{\ell'}^k(r)}$ . If so it sends  $q_k(x_{\ell'}, W_{\ell'}^k(r))$  to all validators in community  $\mathcal{V}_{\ell'}$ . Otherwise,  $m_k$  does not send any data regarding transaction  $x_{\ell'}$ .

*Step 2: Receiving perceptions from other validators:* For each transaction  $x_\ell \in X_r(n)$ , each validator  $m_k \in \mathcal{V}_\ell$  receives  $q_k(x_\ell, W_\ell^k(r))$  that are sent in the Step 1 from all other validators  $m_{k'}$ . We denote the set of all such perceptions by  $Q_k^\ell(r)$ .

#### Stage 2: Update actual belief and making local decision by each validator:

*Step 1: Updating actual belief:* For each transaction  $x_\ell \in X_r(n)$ , each validator  $m_k \in \mathcal{V}_\ell$  updates  $p_k^\ell(r-1)$  using  $Q_k^\ell(r)$  received at Stage 1 to obtain  $p_k^\ell(r)$ . The details can be found in Section III-C.

*Step 2: Making local decisions:* Validator  $m_k$  calculates  $v_k^\ell(r)$  in (3.1).

#### Stage 3: Reach collective decision within each community $\mathcal{V}_\ell$ :

*Step 1: Sending local decision to other validators:* Each validator  $m_k \in \mathcal{V}_\ell$  that has made a decision in Stage 2 sends  $v_k^\ell(r) \in \{0, 1\}$  to all other validators in  $\mathcal{V}_\ell$ .

*Step 2: Receiving local decisions from other validators:* Each validator  $m_k \in \mathcal{V}_\ell$  receives  $v_{k'}^\ell(r)$

from other validators  $m_{k'} \in \mathcal{V}_\ell$ .

*Step 3 (a): Reaching collective decision:* Each validator  $m_k \in \mathcal{V}_\ell$  forms  $\mathcal{M}^\ell(r) = \{m_{k'} \mid m_{k'} \in \mathcal{V}_\ell, v_{k'}^\ell(r') \in \{0, 1\} \text{ for } 1 \leq r' \leq r\}$ , i.e., the set of validators in community  $\mathcal{V}_\ell$  that have made local decisions about the validity of  $x_\ell$  up to round  $r$ . Then validator  $m_k$  checks if there is at least  $\lfloor \frac{|\mathcal{V}_\ell|}{2} \rfloor + 1$  identical local decisions in  $\mathcal{M}^\ell(r)$ . If so, the final decision  $V^\ell$  regarding the validity of transaction  $x_\ell$  will be based on this agreement.

*Step 3 (b): Forcing validators to make a decision:* If there are less than  $\lfloor \frac{|\mathcal{V}_\ell|}{2} \rfloor + 1$  identical decisions in  $\mathcal{M}^\ell(r)$  and if  $r = |W_\ell|$ , then each validator  $m_k \in \mathcal{V}_\ell \setminus \mathcal{M}^\ell(r)$  makes a decision based on the following rule:

$$\eta_1(\mathbf{a}_\ell, \mathbf{y}_k) - p_k^\ell(r = |W_\ell|) \underset{v_k^\ell=1}{\overset{v_k^\ell=0}{\geq}} p_k^\ell(r = |W_\ell|) - \eta_2(\mathbf{a}_\ell, \mathbf{y}_k). \quad (3.4)$$

This forces all validators in community  $\mathcal{V}_\ell$  to make local decisions by round  $r = |W_\ell|$  and the final decision can be obtained using (3.2).

**Stage 4: Broadcasting final decisions:** For each community  $\mathcal{V}_\ell$ , if a final decision  $V^\ell$  on transaction  $x_\ell$  is reached in the current round  $r$ , then  $V^\ell$  is broadcast to all validators in  $X(n)$ . Each validator then obtains  $X_{r+1}(n)$  by removing all transactions whose final decisions have been reached in round  $r$ .

*Remark 1:* Although the final collective decision for transaction  $x_\ell$  must be made by round  $|W_\ell|$ , it can be reached sooner if sufficient number of validators in community  $\mathcal{V}_\ell$  reach local decisions before round  $|W_\ell|$ . Therefore, all processings in epoch  $n$  can be completed before round  $R_n$  if every community reaches a final decision earlier than round  $|W_\ell|$ .

*Remark 2:* Note that if the final decision  $V^\ell$  for transaction  $x_\ell$  is reached in round  $r$ , then in the remaining rounds  $r' > r$  of the epoch validators in  $\mathcal{V}_\ell$  participate only in Step 1 of Stage 1 and Stage 4.

*Remark 3:* The HybridChain can be viewed as a hybrid of the sharded blockchain and the DAG. In particular, it is evident that validators process transactions concurrently in  $\lambda$  communities,

resembling sharded blockchains. Furthermore, akin to DAG distributed ledgers, HybridChain utilizes thresholds as defined in (3.1) to determine the acceptance or rejection of transactions. Additionally, similar to DAG where the relationships among transactions dictate the set of valid transactions, HybridChain assesses the validity of a transaction based on its confliction to the transactions within its witness set.

*Remark 4:* The decentralized nature of HybridChain is apparent in its implementation. Specifically, each transaction  $x_\ell$ , is stored by  $2f + 2$  validators in the set  $\mathcal{V}_\ell$ , which ensures a decentralized ledger. Furthermore, the final decision  $V^\ell$  is not solely determined by local decisions of validators in  $\mathcal{V}_\ell$ , but also influenced by perceptions received from all other validators in different communities  $\mathcal{V}_{\ell'}$  during Stage 1. This decentralized approach to processing  $x_\ell$  ensures that computations related to its validation are decentralized across the network.

### Updating reliability and weight in each epoch $n$

At the end of each epoch  $n$ , the reliability profiles  $\{\rho_1^m, \dots, \rho_M^m, \rho_1^u, \dots, \rho_N^u\}$  of validators and users are updated as follows. For each validator  $m_k$  denote  $\mathcal{Q}_k(n) = \{q_k(x_\ell, x_j) \in \{0, 1\} : x_\ell \in X(n), x_j \in W_\ell\}$  as the set of binary perceptions  $m_k$  sent in epoch  $n$ . And for each pair  $(x_\ell, x_j)$ , denote  $q^*(x_\ell, x_j) \in \{0, 1\}$  as the majority values of all perceptions  $\{q_k(x_\ell, x_j)\}$  sent by validators in epoch  $n$ . Then the reliability of validator  $m_k$  is updated as:

$$\begin{aligned} \rho_k^m(n) &= \zeta_1 \rho_k^m(n-1) \\ &+ \frac{1 - \zeta_1}{|\mathcal{Q}_k(n)|} \sum_{q_k(x_\ell, x_j) \in \mathcal{Q}_k(n)} q^*(x_\ell, x_j) \oplus q_k(x_\ell, x_j). \end{aligned} \quad (3.5)$$

Moreover, let  $X_s(n) \subset X(n)$  be the set of transactions submitted by user  $s$ . If  $X_s(n) \neq \phi$ , then the reliability of user  $s$  is updated as:

$$\rho_s^u(n) = \zeta_2 \rho_s^u(n-1) + \frac{1 - \zeta_2}{|X_s(n)|} \sum_{x_\ell \in X_s(n)} V^\ell, \quad (3.6)$$

Otherwise  $\rho_s^u(n) = \rho_s^u(n-1)$ . In (3.5)-(3.6),  $0 < \zeta_1, \zeta_2 < 1$  are forgetting factors to count for time evolution.

Furthermore, the weight vector  $\mathbf{y}_k$  for each validator  $m_k$  is updated every  $\mathcal{N}$  epochs. Denote  $\mathcal{X}(t) = \bigcup_{n=(t-1)\mathcal{N}+1}^{t\mathcal{N}} X(n)$  as the set of all transactions that are submitted in epochs  $(t-1)\mathcal{N} + 1, \dots, t\mathcal{N}$ . Denote further  $\mathcal{X}_k(t) \subset \mathcal{X}(t)$  such that for each  $x_\ell \in \mathcal{X}_k(t)$ ,  $m_k \in \mathcal{V}_\ell$ . Then the weight vector  $\mathbf{y}_k(t)$  is given by the following optimization problem:

$$\mathbf{y}_k(t) = \arg \max_{\mathbf{y}_k} \left( -\frac{1}{\mathcal{N}} \sum_{x_\ell \in \mathcal{X}_k(t)} \left[ V^\ell \log(\sigma(\mathbf{a}_\ell^T \mathbf{y}_k)) + (1 - V^\ell) \log(1 - \sigma(\mathbf{a}_\ell^T \mathbf{y}_k)) \right] \right) \quad (3.7)$$

that will be used for epochs  $t\mathcal{N} + 1, \dots, (t+1)\mathcal{N}$ . The minimization problem in (3.7) involves the log loss function, which is commonly used for binary classification [99]. In this context,  $\mathbf{a}_\ell$  represents the feature vector, and  $\mathbf{y}_k$  is the classifier. The log loss function quantifies the discrepancy between the predicted probabilities and the true labels, effectively penalizing incorrect predictions and promoting well-calibrated probability estimates. Once the classifier  $\mathbf{y}_k$  has been obtained, it assigns higher scores to transaction  $x_\ell$  (i.e., a larger value of  $\sigma(\mathbf{a}_\ell^T \mathbf{y}_k)$ ) if it resembles valid transactions, and lower scores if it resembles invalid transactions. Validator  $m_k$  utilizes these scores to calibrate the thresholds in (3.1) for accepting or rejecting transactions in subsequent epochs.

### 3.3.4 Updating Rules Inspired by Distributed Learning

In this section, HybridChain's consensus mechanism is dissected to reveal its foundation in distributed learning principles. Central to this mechanism is the validators' belief updating process. A validator  $m_k$ 's decision to accept or reject transaction  $x_\ell$  is based on the actual belief  $p_k^\ell(r)$ , representing the validator's assessment of the transaction's validity at the end of round  $r$  in the set  $X(n)$ , as introduced in (3.1). The section further introduces the concept of intermediate belief  $\psi_k^\ell(r)$ , which informs the actual belief  $p_k^\ell(r)$ . This intermediate belief is constructed from the perceptions  $q_k(x_\ell, x_j)$  exchanged among validators, reflecting the distributed learning approach. The details of how  $p_k^\ell(r)$  is updated, incorporating Bayesian methods and reliability assessments  $\rho_k^m$ , will be

explained later in this section. This process exemplifies how HybridChain leverages distributed learning not only to enhance transaction processing efficiency but also to continuously refine the validators' decision-making capabilities.

## Updating actual beliefs

In the literature on distributed consensus algorithms [100]–[102], the actual belief  $p_k^\ell(r)$  at each round  $r$  can be expressed using the general form  $p_k^\ell(r) = g\left(\{p_{k'}^\ell(r-1)\}_{k' \in \mathcal{M} \cup \{k\}}, \psi_k^\ell(r)\right)$ . Here,  $g$  is called the opinion pooling function that is algorithm-specific, and  $\{p_{k'}^\ell(r-1)\}_{k' \in \mathcal{M} \cup \{k\}}$  is the set of all actual beliefs that validator  $m_k$  receives from other validators  $m_{k'}$ , in addition to its own actual belief. At each round  $r$ , each validator takes into account its intermediate belief and the actual beliefs of other validators from the previous round to determine the actual belief for the current round.

Consensus algorithms, such as those in [103], [104], typically use an opinion pooling function  $g$  that is a linear or log-linear averaging function. However, [105] shows that the min-rule can be more effective in achieving faster convergence in learning the correct actual belief, and better Byzantine resilience. Therefore, in HybridChain, at each round  $r$  of epoch  $n$ , each validator  $m_k$  updates the actual belief  $p_k^\ell(r-1)$  for each transaction  $x_\ell \in X_r(n)$  as follows.

First, validator  $m_k$  sends its actual belief  $p_k^\ell(r-1)$  to all other validators  $m_{k'} \in \mathcal{V}_\ell$  and receives  $p_{k'}^\ell(r-1)$  from all other validators  $m_{k'}$ . Next, it sorts the received values  $p_{k'}^\ell(r-1)$  and removes the  $f$ -highest and  $f$ -lowest values. The set of remaining validators whose actual beliefs are not discarded by validator  $m_k$  at round  $r$  is denoted by  $\mathcal{D}_k^\ell(r)$ . The actual belief  $p_k^\ell(r)$  is then computed as

$$p_k^\ell(r) = \min \left\{ \{p_{k'}^\ell(r-1)\}_{k' \in \mathcal{D}_k^\ell(r) \cup \{k\}}, \psi_k^\ell(r) \right\}. \quad (3.8)$$

*Remark 5:* Suppose  $W_\ell^k(r) = x_j$  and that validator  $m_k \in \mathcal{V}_\ell$  and  $m_k \in \mathcal{V}_j$ . Then in case  $q_k(x_\ell, x_j) = 0$ , instead of updating  $p_k^\ell(r)$  in Stage 2, validator  $m_k$  sets  $p_k^\ell(r) = 0$  and follows the rest of the processing procedure as usual.

*Remark 6:* The rationale behind selecting  $\lambda = \lfloor M/(2f+2) \rfloor$  is to ensure that, in (3.8), after

removing the  $f$ -highest and  $f$ -lowest values of  $p_{k'}^\ell(r-1)$  by validator  $m_k$ , at least one validator's opinion other than  $m_k$  remains. This condition can be satisfied by having a minimum of  $2f+2$  validators store each transaction, so that in the worst case even if both  $m_k \in \mathcal{V}_\ell$  and  $m_k \in \mathcal{V}_j$  for some  $x_j \in W_\ell$ , at least one other validator's opinion is incorporated.

### Updating intermediate beliefs

To determine  $p_k^\ell(r)$ , as shown in (3.8), validator  $m_k \in \mathcal{V}_\ell$  must first obtain its intermediate belief  $\psi_k^\ell(r)$ . As previously stated,  $\psi_k^\ell(r)$  is computed based on the perceptions  $q_{k'}$  that other validators  $m_{k'}$  send to validator  $m_k$ . To be more specific, let  $Q_k^\ell(r)$  denote the set of all perceptions  $q_{k'}(x_\ell, W_\ell^{k'}(r))$  received by validator  $m_k$  during Step 2 of Stage 1. To determine  $\psi_k^\ell(r)$  in round  $r$ , validator  $m_k$  applies the Bayesian rule as in (3.9).

$$\psi_k^\ell(r) = \frac{P(Q_k^\ell(r) \mid x_\ell \text{ is valid}) \psi_k^\ell(r-1)}{P(Q_k^\ell(r) \mid x_\ell \text{ is valid}) \psi_k^\ell(r-1) + P(Q_k^\ell(r) \mid x_\ell \text{ is invalid}) (1 - \psi_k^\ell(r-1))}. \quad (3.9)$$

The conditional probabilities in (3.9) are computed using the reliabilities  $\rho_{k'}^m$  of the validators  $m_{k'}$  who contributed the perceptions  $q_{m_{k'}}$  in  $Q_k^\ell(r)$  as follows:

$$P(Q_k^\ell(r) \mid x_\ell \text{ is valid}) = \prod_{q_{k'} \in Q_k^\ell(r)} \left[ \rho_{k'}^m \mathbf{1}_{[q_{k'}=1]} + (1 - \rho_{k'}^m) \mathbf{1}_{[q_{k'}=0]} \right], \quad (3.10a)$$

$$P(Q_k^\ell(r) \mid x_\ell \text{ is invalid}) = \prod_{q_{k'} \in Q_k^\ell(r)} \left[ ((1 - \beta) \rho_{k'}^m + \beta(1 - \rho_{k'}^m)) \mathbf{1}_{[q_{k'}=1]} + (\beta \rho_{k'}^m + (1 - \beta)(1 - \rho_{k'}^m)) \mathbf{1}_{[q_{k'}=0]} \right]. \quad (3.10b)$$

where  $\beta = \frac{2^{|W_\ell|-1}}{2^{|W_\ell|}-1}$  is the probability of  $x_\ell$  being in conflict with a given  $x_j \in W_\ell$  given that  $x_\ell$  is an invalid transaction. The denominator of  $\beta$  is the total number of possible perceptions of the conflict of  $x_\ell$  with respect to transactions in  $W_\ell$ , excluding the case that it is not in conflict with any in  $W_\ell$ ; the numerator of  $\beta$  is the total number of perceptions given  $x_\ell$  is in conflict with  $x_j$ .

Finally, it should be mentioned that validators use  $p_k^\ell(0) = \rho_s^u$  and  $\psi_k^\ell(0) = \rho_s^u$  as initial values in (3.8) and (3.9), where  $\rho_s^u$  is the reliability of user  $s$  that sends transaction  $x_\ell$ .

### 3.4 Security Analysis and Comparison

In this and next sections, we will conduct a comparative analysis between HybridChain and two other notable schemes in the field of distributed ledgers: IOTA, a benchmark DAG scheme, and Omniledger, a sharded blockchain. The primary objective of this section is to evaluate security features of all three schemes.

#### 3.4.1 Replay Attack

A replay attack [106] occurs when an attacker intercepts a transaction and resends it at a later time, in an attempt to deceive the recipient into thinking that a new transaction is taking place, when in fact it is just a repeat of a previous transaction. Replay attacks can be particularly problematic in multi-party transactions, where all parties must agree to commit to a certain action simultaneously, or else the entire transaction is rolled back. In this scenario, a replay attack could cause incorrect decisions to be made about the transaction, leading to fraud, loss of funds, or other security breaches.

IOTA Tangle is resilient against replay attacks. Transactions in IOTA Tangle are interlinked, forming a “tangle” that undergoes validation by network nodes, which ensures the uniqueness of each transaction and prohibits replication. Furthermore, every transaction in Tangle requires a distinct identifier called a “branch” and a “trunk,” pointing to two previous transactions within the Tangle. When a transaction is broadcast to the IOTA network, it undergoes validation by the network nodes. If a node receives a transaction with a branch and trunk that have already been utilized, the transaction is rejected, effectively thwarting any replay attempts.

The Omniledger is vulnerable to replay attacks [107], as it utilizes Atomic commitment for transaction processing. A potential attacker can easily execute a replay attack by intercepting a cross-shard transaction, and then resending it to the involved shards. Nodes within the network are unable to differentiate between honest but delayed transactions and maliciously replayed ones,

resulting in conflicting views among different shards regarding the transactions in question. This is particularly concerning for cross-shard transactions, as validators in each shard lock the transactions until a decision is made to either commit or abort the transaction. In the case of a replay attack, the transaction may become locked indefinitely, or even result in a double-spending scenario.

In contrast to Omniledger, HybridChain is resilient against replay attacks due to its unique design. While similar to sharded blockchains in the sense that our scheme processes transactions in parallel, it is not a sharded blockchain itself. In contrast to sharded systems where only a subset of nodes process transactions, non-sharded blockchains require consensus from all nodes for each transaction, ensuring that it is unique and cannot be replayed at a later time. Specifically, for each transaction  $x_\ell \in X(n)$ , the set of validators in  $\mathcal{V}_j$  for each transaction  $x_j \in W_\ell$  are randomly assigned and can be entirely different. This means that each validator  $m_k \in \mathcal{V}_\ell$  receives a set of perceptions  $Q_k^\ell(r)$  from all other validators in the network, making each validator a participant in the verification of transaction  $x_\ell$ .

### 3.4.2 Message Withholding Attack

A message withholding attack [108] involves a malicious actor deliberately withholding important messages or transactions that are necessary for the system to function correctly. In particular, the attacker can concentrate on a particular transaction or set of transactions, and prevent the circulation of messages required to authenticate them. As a result, the transaction may not be included in a block, or its confirmation could be postponed, causing disruptions in the normal functioning of the system. Message withholding attacks are particularly potent in systems that employ a consensus mechanism based on a quorum to approve transactions. Through the suppression of messages, the attacker can impede the system's ability to reach consensus and even cause a fork in the blockchain or ledger.

In IOTA's Tangle, the act of withholding a single message does not necessarily hinder the confirmation of a transaction since the confirmation process does not depend solely on a linear sequence of transactions. Rather, transactions are verified based on a consensus mechanism that

considers the entire network of interconnected transactions. This distinctive architecture and consensus mechanism of IOTA make it resilient to message withholding attacks.

On the other hand, Omniledger is susceptible to the message withholding attack. Specifically, for intra-shard transactions, the shard leader, responsible for processing the transaction, receives the transaction and dispatches it to the validators within the shard. If the shard leader is malicious and launches a message withholding attack, it can prevent the transaction from being processed. Also, in the case of cross-shard transactions, an adversary can create a cross-shard transaction and send the transaction to the source shard but withhold the transaction from the destination shard causing the cross-shard transaction to remain active and keeping the validators occupied with the transaction [107].

HybridChain is resilient to message withholding attacks. To process a transaction  $x_\ell$ , each of the witness transactions  $x_j \in W_\ell$  is stored by  $2f + 2$  validators, assuming that the network has no more than  $f$  dishonest validators. Validator  $m_k \in \mathcal{V}_\ell$  receives perceptions  $q_{m_{k'}}(x_\ell, x_j)$  from validators  $m_{k'} \in \mathcal{V}_j$  for all transactions  $x_j \in W_\ell$ . Hence, in the event of a message withholding attack by dishonest validators, at least  $f + 2$  other validators exist to inform validator  $m_k$  of the transaction's validity with respect to each witness transaction.

### 3.4.3 Orphanage Attack

In a distributed ledger scheme, when a conflict occurs between a new block or transaction and an existing one, the network must choose to orphan one of them. If the scheme employs Nakamoto consensus, where the validity of blocks or transactions is determined by the ledger structure, selecting the new block or transaction results in the old one being orphaned, wasting the efforts put into its creation. This problem becomes more severe when a subsequent block or transaction conflicts with the chosen block or transaction. If the network selects the subsequent block or transaction, it not only orphans the old block or transaction but also orphans the one it was chosen over.

The orphanage attack [109] involves taking advantage of the orphanage problem and create transactions or blocks to make the existing ones to be left behind. By conducting simulations and

analyzing particular strategies, [109] revealed that the impact of the orphanage attack could be significant in IOTA, to the extent that a sufficiently powerful attacker could cause 96% of transactions in IOTA to be orphaned.

On the other hand, for schemes that do not employ Nakamoto consensus, if a conflict arises between a new transaction or block and an existing one, the network keeps the existing one and discards the new one. The Omniledger uses Fast Byzantine Consensus (FBC) as the consensus algorithm, which is a classical consensus. FBC can resist orphanage attacks by providing immediate finality for confirmed blocks, employing a synchronized confirmation process to prevent manipulation. These features ensure that once a block is confirmed by the network, it cannot be discarded or orphaned.

Similar to Omniledger, in HybridChain all submitted transactions are placed in a queue and processed in batches in parallel during epochs, ensuring that none of the transactions are orphaned due to acceptance of another transaction. Thus, HybridChain is immune to the orphanage attack.

#### 3.4.4 Routing Attack

The Border Gateway Protocol (BGP) is a protocol used by Autonomous Systems (AS), which are groups of interconnected Internet Protocol (IP) routing prefixes controlled by one or more network operators, to share information about subnet reachability through advertisements. BGP does not verify the accuracy of advertised routes, making it possible for a malicious AS to add and propagate advertisements for routes that do not actually exist. BGP hijacking occurs when an AS advertises a route that it does not have in order to attract Internet traffic intended for that destination.

In IOTA, a full node must connect with random full nodes to maintain up-to-date information. This randomness is critical to prevent adversaries from compelling specific nodes to become peers, and to prevent predictable network configurations that could result in eclipse attacks. In an eclipse attack, a group of colluding nodes surrounds a target node and intercepts all communication with the rest of the network, potentially causing ledger consensus issues such as double spending or forking. In [110] it is demonstrated that BGP hijacking may be successful in the IOTA network unless two

full nodes are in the same AS and use intra-AS routes. However, forcing nodes to be neighbors in the same AS would reduce the network’s overall connectivity, which is undesirable.

On the other hand, in contrast to IOTA, Omniledger is immune to such routing attack. This is because in Omniledger, all nodes hold a partial copy of the ledger, and the processing of transactions is distributed among all nodes. To prevent routing attacks, OmniLedger uses a hybrid routing approach that combines both centralized and decentralized routing mechanisms. In this approach, the routing information is divided into two parts: a centralized directory that contains the IP addresses and public keys of all the shards, and a decentralized routing table that contains the routing information for each shard. The centralized directory helps to prevent attacks on the routing information, while the decentralized routing table helps to prevent attacks on the transaction history of each shard.

Moreover, HybridChain achieves a well-distributed computation of transaction processing. Similar to Omniledger, it avoids relying on centralized entities such as full nodes. Instead it allows all validators to access the necessary information by distributing the ledger data. Thus, the routing attack cannot compromise the consistency or security of the ledger data. In particular, even if the attacker targets validator  $m_k$ , the decision  $v_k^\ell$  and the final decision  $V^\ell$  regarding transaction  $x_\ell$  are unlikely to be affected. This is because the calculation of  $p_k^\ell(r)$ , which determines  $v_k^\ell$ , depends on all perceptions in  $Q_k^\ell(r)$  for each round  $r$ , where  $Q_k^\ell(r)$  is a group of perceptions obtained from various validators. Therefore, in order to change  $v_k^\ell$ , the attacker must disrupt many connections between validator  $m_k$  and other validators, making it difficult to influence the decision-making process of any validator, and consequently the final decision.

Finally, we summarize whether or not each distributed ledger scheme is resistant to various attacks in Table 3.3.

Attack Type	IOTA	Omniledger	HybridChain
Replay Attack	Yes	No	Yes
Message Withholding Attack	Yes	No	Yes
Orphanage Attack	No	Yes	Yes
Routing Attack	No	Yes	Yes

Table 3.3: Resistance to various attacks by different distributed ledger schemes.

### 3.5 Simulation Results

This section provides a thorough comparison of HybridChain with IOTA and Omniledger, focusing on the system’s latency, throughput, and scalability, to showcase the advantages of HybridChain. Additionally, we assess how our decentralized scheme compares with centralized classification algorithms, that are commonly used for detecting fraudulent transactions.

#### 3.5.1 Simulation Setups

##### IOTA

To simulate IOTA, we utilized the simulator TangleSim [111] along with certain software modifications and measurement tools found in [112] and [113], which simulate an improved version of the IOTA [95], where the nodes in the network use the Approval Weight (AW) mechanism to reach consensus about the set of the valid transactions.

Specifically, when a light node intends to attach a transaction to the Tangle it must select  $\mathcal{P} \geq 2$  older transactions using the Restricted Uniform Random Tip Selection (R-URTS) protocol, which chooses parents uniformly at random from the pool of tip transactions whose age are less than a threshold  $\delta$ . Then to reach consensus about the set of valid transactions, nodes first calculate the *Mana* score for all participant nodes based on their ownership of a scarce resource, which represents their influence. Then using the Mana score, nodes determine the set of valid transaction in the Tangle by computing the *cumulative weight* of all transactions and comparing it with a predefined threshold  $\theta$ . If the cumulative weight of a transaction exceeds  $\theta$ , it is considered valid.

In the IOTA simulations, honest nodes select parents following the R-URTS protocol and issue valid transactions. The adversary nodes always issue invalid transactions; and while choosing a parent they select the one with the highest possible age; and if possible they try to select their own transactions.

Denote the rate of transaction issuance per minute by  $\gamma = \gamma_a + \gamma_h$ , where  $\gamma_a$  and  $\gamma_h$  are transaction issuance rates of adversary nodes and honest nodes, respectively. Define  $S = \frac{\gamma_a}{\gamma}$  as the spamming

rate of the adversary nodes. Each transaction must select  $\mathcal{P} = 2$  other transactions as parents with an age restriction of  $\delta = 1$  minute. Also the confirmation threshold  $\theta = 0.66(1 - \tau)$  where  $\tau$  is the percentage of adversary nodes. We set the number of nodes in IOTA as  $M = 1000$ . In each simulation, we execute the software for a duration of 5 minutes, generating  $5\gamma$  transactions at various transaction issuance rates denoted by  $\gamma$ . These transactions are then attached to the Tangle, forming an integral part of the simulation process.

## Omniledger

To simulate Omniledger, we utilized OverSim [114], a modular framework, on OMNeT++ 6.0 [115], a process-based discrete-event simulator. The simulation parameters are set based on [5] as follows. The required percentage of votes for accepting a transaction, as well as other consensus-related activities like leader selection, is set to 67%. Additionally, the block size is fixed at 1 MB. The task completion and voting timeout values are set to 500 milliseconds, and an epoch transition occurs when all tasks are completed, and the network has processed the entire task list. For all simulations, the bandwidth of communication links and latency between validators are set to 20 Mbps, and 100 ms, respectively.

To generate transactions, we employed Bitcoin Core [116] version 23.1, which was the latest release at the time of writing this chapter. The rate of incoming transactions in Omniledger is represented by  $\gamma$ , and the total number of generated transactions in all simulations matched that in IOTA, which is  $5\gamma$ .

In Omniledger, it is important to note that intra-shard and cross-shard transactions are processed separately, with cross-shard transactions generally taking longer to process than intra-shard ones. The number of shards in Omniledger, denoted as  $\Gamma$ , varied across different simulations. We provide the specific value of  $\Gamma$  for each simulation. Based on [117], using the RandHound algorithm in Omniledger with  $\Gamma = 4$ , approximately 81% of transactions are categorized as cross-shard, while with  $\Gamma = 16$ , nearly 95% of transactions are classified as cross-shard. Thus, the performance of Omniledger is expected to be heavily influenced by the value of  $\Gamma$ .

In our simulations, we considered  $M = 1000$  validators in Omniledger. The percentage of Byzantine dishonest validators, denoted as  $\tau$ , was also specified for each simulation, similar to IOTA. A Byzantine dishonest validator adheres to the network’s protocol but intentionally casts incorrect votes for both intra-shard and cross-shard transactions, whereas an honest validator always votes correctly.

## **HybridChain**

To simulate HybridChain, which shares similarities with Omniledger, we employed OverSim as our simulation platform. However, to customize and implement our scheme, we made significant modifications to four key modules of Omniledger.

Firstly, we modified the validator module, which represents a validator in the blockchain network. Our modification accounted for the random grouping of validators in each epoch and included operations at all of Stages described in Section III-C, and managing validators’ interactions during each round.

Secondly, we made changes to the consensus module responsible for achieving consensus among validators regarding the set of valid transactions. These modifications encompassed adjusting the voting and decision-making logic to incorporate HybridChain’s consensus mechanism detailed in Section III.

Thirdly, we modified the epoch module, which represents the epochs in HybridChain comprising multiple rounds. We redefined the structure and duration of each epoch, as well as the transitions between epochs. Similar to Omniledger, we set the task completion and voting timeout values to 500 milliseconds, with a round transition occurring when all tasks are completed, and the network has processed the entire task list. We also set the bandwidth 20 Mbps and imposed a latency of 100 ms on all communication links in all simulations.

Lastly, we adapted the transaction module to accommodate the specific transaction structure and validation rules of HybridChain. Specifically, in each simulation, similar to the corresponding simulations for IOTA and Omniledger, we generated the same number of  $5\gamma$  transactions, consisting

of both valid and invalid transactions. Additionally, for all simulations, we utilized the probability distributions listed in Table 3.4 to generate the attribute vector  $\mathbf{a}_\ell$  for valid or invalid transactions  $x_\ell$ . Here, it should be noted that the values of the elements in the attribute vector are truncated, to certain ranges as indicated in the last column.

Attribute Vector Element	Valid	Invalid	Range
$a_\ell[1]$	Gamma(3, 1)	Gamma(17.5, 0.5)	(0, 10]
$a_\ell[2]$	Norm(0.5, 0.15)	Norm(0.25, 0.075)	(0, $+\infty$ )
$a_\ell[3]$	Exp(10)	Exp(5)	(0, 50]
$a_\ell[4]$	Pois(1.5)	Pois(2.5)	[1, $+\infty$ )
$a_\ell[5]$	Norm(0.7, 0.1)	Norm(0.4, 0.1)	(0, 1]

Table 3.4: Distribution of attribute vector elements for generating valid and invalid transactions. For  $a_\ell[3]$ , the generated number is rounded down to find a discrete number of rounds.

To simulate an invalid transactions  $x_\ell$ , after determining the size  $W_\ell$  using the distribution of  $a_\ell[4]$  from Table 3.4, we randomly select transactions  $x_j \in W_\ell$  to be in conflict with  $x_\ell$ .

Also, in (3.1) we use the sigmoid function  $\sigma(\mathbf{a}_\ell^T \mathbf{y}_k) = \frac{1}{1+e^{-\mathbf{a}_\ell^T \mathbf{y}_k}}$  and set  $\mu_1 = 1$  and  $\mu_2 = 0.5$ , which results in  $0.5 \leq \eta_1(\mathbf{a}_\ell, \mathbf{y}_k) \leq 1$  and  $0 \leq \eta_2(\mathbf{a}_\ell, \mathbf{y}_k) \leq 0.5$ . Moreover, we set the number of users  $N = 100$  and updating frequency  $\mathcal{N} = 20$  epochs. Furthermore, for the forgetting factors in (3.5)-(3.6), we set  $\zeta_1 = 0.98$ ,  $\zeta_2 = 0.9$ . Additionally, the reliability of validators and users are uniformly generated in range [0.3, 0.8].

To initially train the weight vector  $\mathbf{y}_k$ , we use 10000 transactions, with half labeled as valid and half as invalid. We utilize the Scikit-learn [118] library in Python to solve the minimization problem in (3.7).

Similar to IOTA and Omniledger, the number of validators is set as  $M = 1000$ , the percentage of dishonest validators is denoted by  $\tau$ . In HybridChain, honest validators always send correct perceptions to other validators and make local decisions following the defined rules in Section III. On the other hand, dishonest validators consistently send incorrect perceptions to other validators and make incorrect local decisions.

### 3.5.2 Results

#### **Classification**

In order to evaluate the performance of our decentralized consensus scheme in classifying the validity of transactions, we compared it to a centralized classification algorithm, the Random Forest algorithm [119], known for its high accuracy. To conduct the simulation, we generated a dataset of 10000 transactions, which were used to train a Random Forest classifier consisting of 100 trees. Each tree utilized a random subset of two or three attributes from the five features in the attribute vector. Then, we evaluated the performance of the trained Random Forest model on a separate set of 5000 transactions, comprising an equal number of valid and invalid transactions, yielding an accuracy of 98.2%.

We subjected the same set of 5000 transactions to HybridChain and a network with varying numbers of validators and different values of  $\tau$ . The results are depicted in Fig.3.2, demonstrating that HybridChain consistently exhibits stable performance, under different number of dishonest validators or different network sizes. This remarkable performance can be attributed to the collaborative decision-making mechanism of HybridChain, which aids validators in enhancing the accuracy of their decisions.

#### **Latency**

We define latency the amount of time it takes for a transaction to be confirmed and added to the ledger. It is the delay between the moment a transaction is initiated and the moment it is considered final and irrevocable.

Fig.3.3 presents the cumulative distributions of latencies of different systems with  $\tau = 20\%$  dishonest validators. For IOTA we set the spamming rates to  $S = 30\%$  and  $S = 70\%$  and set  $\gamma = 3 \times 10^4$ . For Omniledger we set  $\Gamma = 4$  and  $\Gamma = 16$  shards. The results demonstrate that Omniledger has the highest latency for both sharding sizes and IOTA with a spamming rate of  $S = 30\%$  has the lowest latency. The latency of HybridChain is slightly lower than that of IOTA

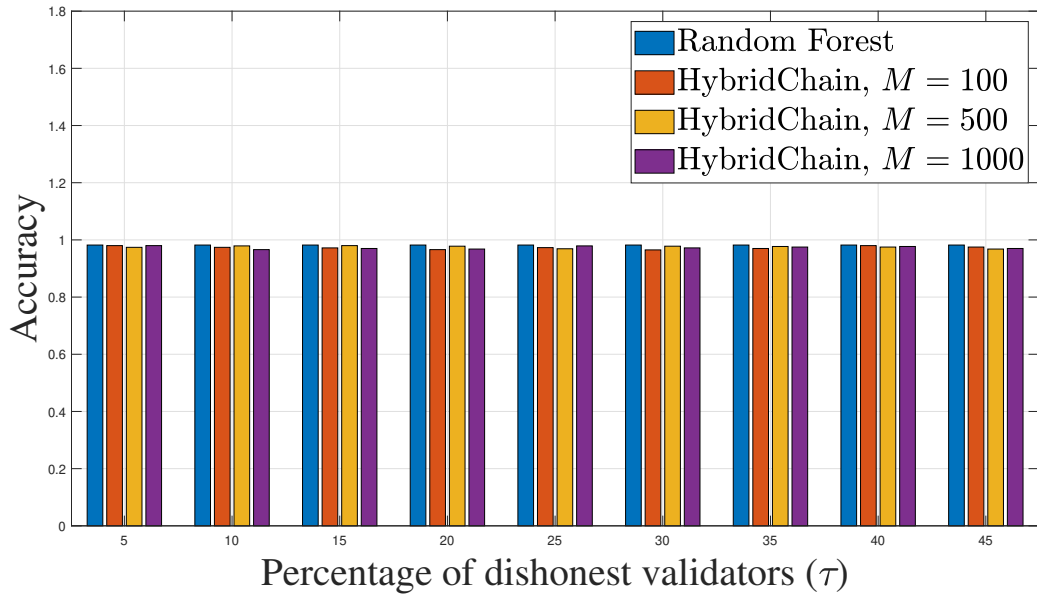


Figure 3.2: Accuracy of transaction validity classification under different values of dishonest validators ( $\tau$ ) and number of validators  $M$  for HybridChain compared to Random Forest.

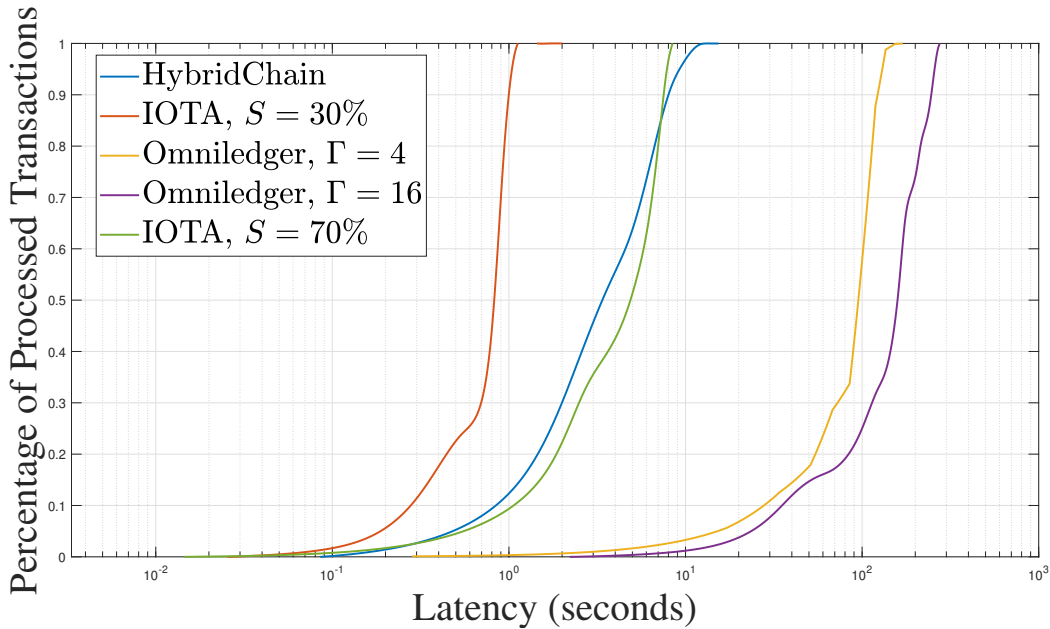


Figure 3.3: Latency distribution with incoming transactions rate  $\gamma = 3 \times 10^4$ ,  $M = 1000$  validators, and  $\tau = 20\%$ .

with a spamming rate of  $S = 70\%$ . However, as illustrated in Fig.3.6 in Section V-B-4, the quick processing of IOTA with a spamming rate of  $S = 30\%$  comes at the expense of reduced verification accuracy.

## Throughput

The throughput is the average number of transactions processed by a scheme per unit of time. However, when comparing two schemes with equal throughput, the one that achieves higher transaction validation accuracy outperforms the other. Therefore, when evaluating the throughput, it is essential to consider two factors: accuracy, which indicates the correctness of decisions made during transaction processing, and the time required for processing transactions. To assess the throughputs of the three schemes, we conducted two simulations.

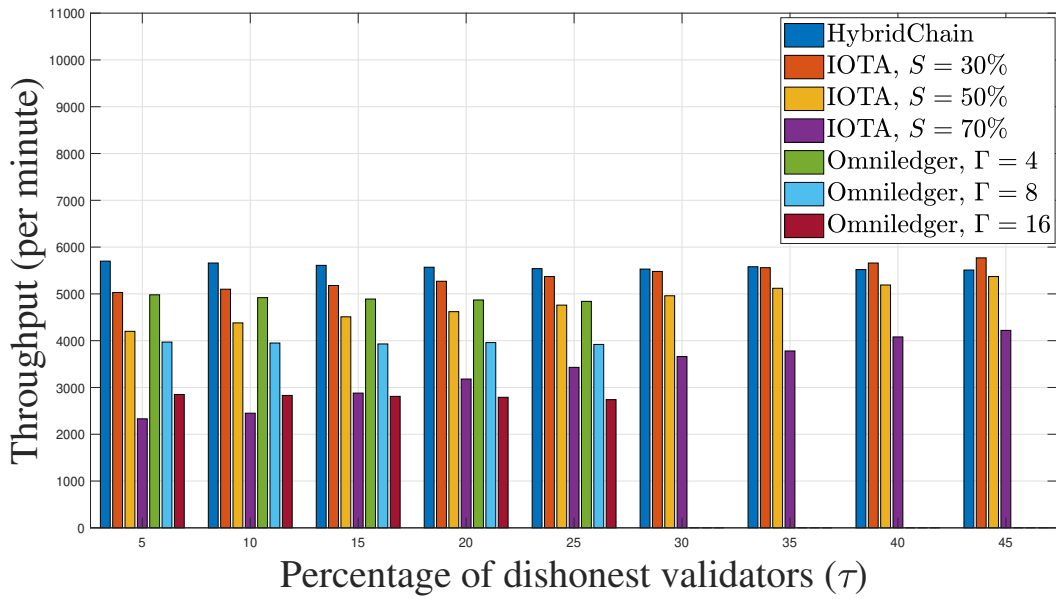
We set the incoming transaction rate  $\gamma = 6000$ . The three schemes were simulated for various values of  $\tau$ , ranging from 5% to 45%. IOTA was simulated with three spamming rates  $S \in \{30\%, 50\%, 70\%\}$ . Omniledger was simulated with three values of  $\Gamma \in \{4, 8, 16\}$ , representing the number of shards in the network. In IOTA, a transaction was considered processed when its cumulative weight exceeds  $\theta$ . In Omniledger and HybridChain, a transaction was considered processed when it was either accepted or rejected by the network consensus. A one-minute time window was used to process transactions for all three schemes, with the number of processed transactions and accuracy recorded.

The results averaged over ten simulation runs, are shown in Fig. 3.4. It is evident that the spamming rate of adversaries heavily influences the throughput and accuracy of IOTA. Specifically, each adversary node impacts IOTA in two ways: first, by increasing the number of orphaned transactions, and second, by confirming invalid transactions. Consequently, when the spamming rate  $S$  increases while keeping  $\tau$  fixed, both the throughput and accuracy decrease. This outcome is anticipated, as stated in [109], due to the existence of a critical spamming rate denoted as  $S_{critical} = \frac{p-1}{p}$ . When  $S$  exceeds  $S_{critical}$ , the inflation of the tip pool size accelerates, resulting in a higher number of orphaned transactions. Also, with an increase in the spamming rate  $S$  of adversaries, more cumulative weights of invalid transactions surpass the threshold  $\theta$ . This occurs because adversary nodes, deviating from the R-URTS protocol, tend to select invalid transactions as parents for their submitted transactions, resulting in a decline in the accuracy of processed transactions.

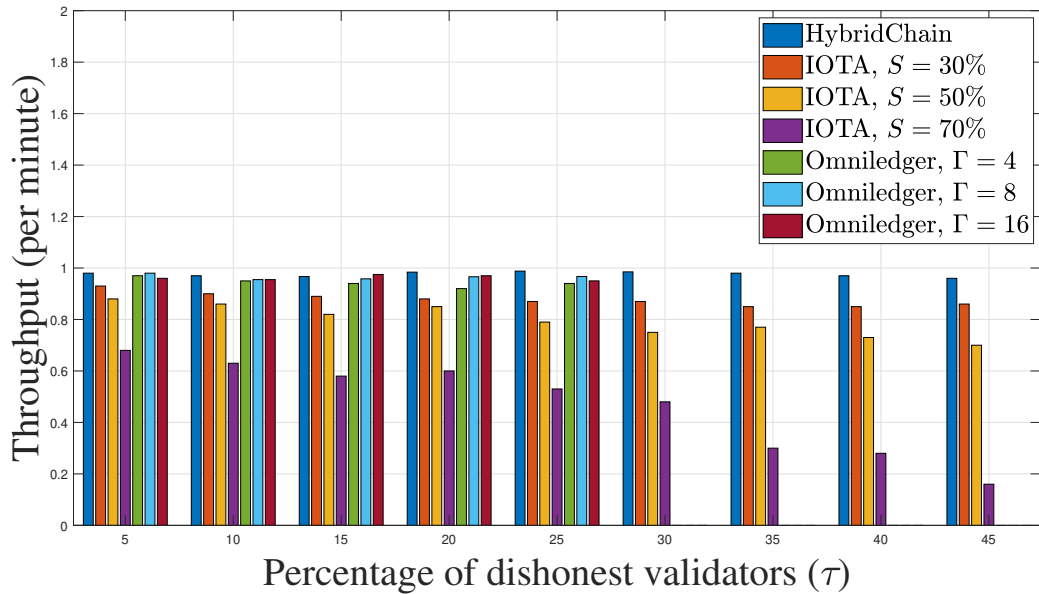
Moreover, when  $S$  is fixed, an increase in  $\tau$  leads to an increase in throughput. This is because the threshold  $\theta$  decreases as a result, allowing a larger number of transactions to be processed. However, the accuracy decreases since a larger number of invalid transactions are validated. The objective of setting the acceptance threshold  $\theta$  as a function of  $\tau$  is to simulate the Mana score in IOTA. Unlike Omniledger and HybridChain, where the number of adversary nodes affects the consensus results, IOTA's performance is unaffected by the quantity of adversary nodes. Instead, it is the distribution of the Mana scores of the nodes that influences the validity of transactions. Therefore, to facilitate a comparison between IOTA and the other two schemes for the same value of  $\tau$ , we consider  $\theta$  as a function of  $\tau$  to simulate the power of adversaries in the network.

For Omniledger, the results showed that for  $5\% \leq \tau \leq 25\%$ , increasing  $\Gamma$  from 8 to 16 led to reduced throughput. This reduction can be attributed to the increased number of cross-shard transactions, which subsequently prolongs the processing time. However, Omniledger demonstrated stable accuracy in its processed transactions. Moreover, with a fixed  $\Gamma$ , an increase in  $\tau$  slightly decreased the throughput due to the additional communication rounds required to achieve consensus using the FBC protocol. The increase in the number of adversary validators increases the time required for consensus by the FBC protocol. However, the accuracy of the processed transactions remained stable, thanks to Omniledger's design that involves validators in transaction processing. On the other hand, for  $25\% \leq \tau \leq 45\%$ , the throughput dropped to almost zero for all  $\Gamma$  values. This indicates Omniledger's inability to handle a network with more than 25% dishonest validators.

As for HybridChain, its throughput is comparable to that of IOTA with  $S = 30\%$  and  $\tau = 45\%$ . However, when  $S = 50\%$  and  $S = 70\%$ , as well as in Omniledger for all values of  $\Gamma$  and for all values of  $\tau$ , HybridChain performs significantly better. The reason is that HybridChain, like DAG schemes, accelerates transaction processing through partial verification. Moreover, as  $\tau$  increases, the throughput of HybridChain slightly decreases since validators require more rounds in each epoch to reach consensus, thereby taking longer to process the transactions. Nevertheless, similar to the findings in Section V-B-1, an increase in  $\tau$  does not impact the accuracy of HybridChain, as it distributes the relevant witness transactions for each transaction  $x_\ell \in X(n)$  in an epoch among the



(a) Throughput



(b) Accuracy

Figure 3.4: Throughput and accuracy based on percentage of dishonest validators, with incoming transactions rate  $\gamma = 6000$  and  $M = 1000$  validators.

validators  $\mathcal{V}_j$  for transactions  $x_j \in W_\ell$ , in a manner that is impervious to the rise in the number of dishonest validators.

In a DAG distributed ledger system, reducing the volume of transactions can increase its

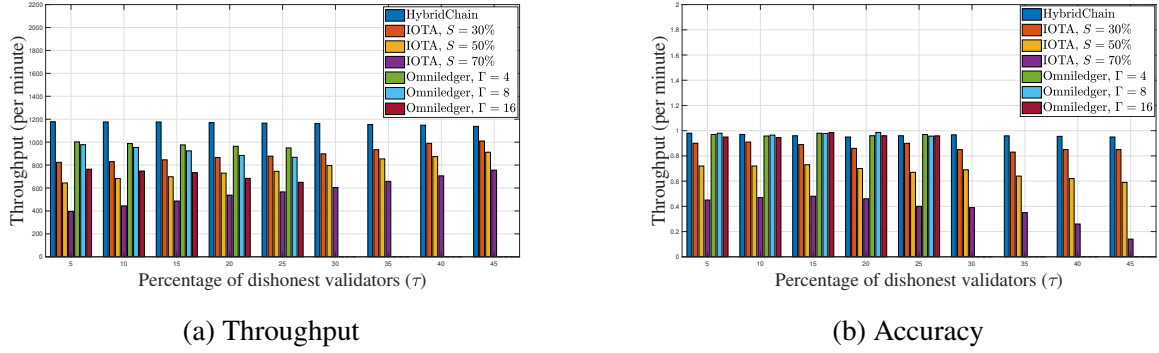


Figure 3.5: Throughput and accuracy based on percentage of dishonest validators, with incoming transactions rate  $\gamma = 1200$  and  $M = 1000$  validators.

susceptibility to attacks, as indicated by [120]. To further investigate this phenomenon, we conducted a second simulation, keeping all settings identical to the first simulation but varying the rate of incoming transactions by setting  $\gamma = 1200$ . This allowed us to observe the effects of low transaction volumes on throughput and accuracy.

The results, shown in Fig. 3.5, demonstrated that a decrease in incoming transactions had a more pronounced impact on IOTA, making it more vulnerable to adversary attacks. This vulnerability stems from the nature of DAG distributed ledgers, where transaction confirmation and validation rely on the structure of the DAG and other transactions. Insufficient transaction volume leads to system instability, resulting in incomplete transaction validation. Consequently, accuracy is reduced, as a greater number of valid transactions remain in the tip pool for longer periods, taking more time to be processed and becoming more susceptible to attacks. Adversaries can exploit this situation to prioritize their own transactions, hindering the processing of valid transactions and further diminishing accuracy. Specifically, by comparing the throughput and accuracy values for  $\gamma = 6000$  and  $\gamma = 1200$ , it becomes evident that the trends observed for  $\gamma = 6000$ , such as increased throughput and reduced accuracy for fixed  $S$  and increased  $\tau$ , persist for  $\gamma = 1200$ . However, the reduction of accuracy is more severe when  $\gamma = 1200$  is considered.

On the other hand, in HybridChain and Omniledger one transaction does not determine the validity of another transaction directly but, therefore changing the transaction volume does not have a significant effect on the throughput and the accuracy.

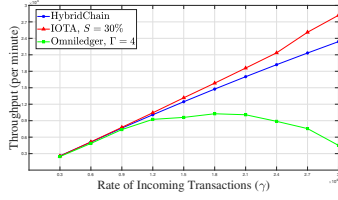
Hence the above results demonstrate that DAG distributed ledger systems, which do not rely on validators to process transactions, can achieve high transaction speeds, allowing network nodes to reach consensus quickly, even for large transaction volumes. However, this speed may come at the cost of consensus accuracy, particularly when facing a powerful adversary who can strategically disrupt the network.

## Scalability

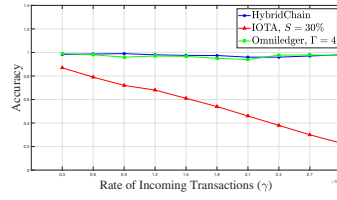
The primary objective of our last simulation is to evaluate the scalability of three schemes, a crucial element within the trilemma triangle. Scalability holds significant importance in the field of distributed ledger technologies, particularly with the proliferation of applications built on distributed platforms and the subsequent surge in user numbers and incoming transactions. In the context of distributed ledgers, scalability pertains to the system's capacity to efficiently manage an increasing volume of transactions, while sustaining high throughput and low latency. Essentially, a distributed ledger system is deemed scalable when it can process a substantial number of transactions without compromising transaction processing speed.

To evaluate the scalability of the three schemes, we used the settings described in Section V-B-3 while varying the rate of incoming transactions  $\gamma$ , from 6000 to  $3 \times 10^4$  with increments of 3000. For each  $\gamma$  value, we measured the throughput of the three schemes and the maximum latency of the processed transactions. In all three schemes, a dishonest fraction of  $\tau = 20\%$  was incorporated. In the case of IOTA, the spamming rate  $S$  was set to 30%, while Omniledger maintained a fixed value of  $\Gamma = 4$  shards. These specific parameter values were chosen based on their demonstrated superior performance in the corresponding scheme simulations conducted in Sections V-B-2 and V-B-3. The objective was to compare the scalability of the other two schemes, IOTA and Omniledger, with HybridChain under their optimal settings.

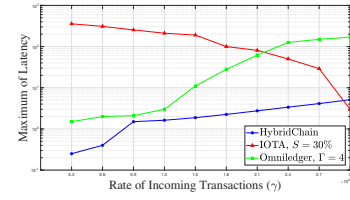
Fig. 3.6 provides a visual representation of the throughput of the three schemes, along with the maximum latency of the processed transactions. Notably, the latency trends differ among the three schemes as the rate of incoming transactions  $\gamma$  increases. In IOTA, the maximum latency decreases



(a) Throughput



(b) Accuracy



(c) Maximum of Latency

Figure 3.6: Scalability based on the rate of incoming transactions ( $\gamma$ ), with  $\tau = 20\%$ , and  $M = 1000$ .

with higher values of  $\gamma$ , while in the other two schemes, the maximum latency increases. This observation highlights a key characteristic of DAG schemes, where the validation of transactions relies on their relationships within the system, rather than on validators. Consequently, an increase in the rate of incoming transactions facilitates faster processing and improves throughput. However, this can potentially lead to decreased accuracy. As depicted in Fig. 3.6, an increase in the rate of incoming transactions  $\gamma$  is accompanied by a reduction in accuracy. The absence of validators to validate transactions can indeed help reduce latency and increase throughput. However, it can also result in erroneous verifications, compromising accuracy.

As expected, Omniledger and HybridChain exhibit the typical behavior of non-DAG schemes, where an increase in the rate of incoming transactions  $\gamma$  leads to higher maximum latency and decreased throughput due to transaction processing and communication among validators. However, the results indicate that Omniledger is more susceptible to the impact of increased  $\gamma$  compared to HybridChain, as evidenced by a larger increase in maximum latency in Omniledger. HybridChain demonstrates excellent scalability, by maintaining nearly constant throughput and exhibiting only a marginal increase in maximum latency, while achieving high accuracy.

### 3.6 Conclusions

The proposed consensus scheme of HybridChain combines the strengths of blockchain and DAG, and by incorporating a decentralized learning algorithm that exploits transaction attributes, achieves fast, accurate, and secure transaction processing. HybridChain can successfully mitigate

various security attacks by decentralizing storage and computations, whereas IOTA and Omniledger exhibit vulnerabilities to these attacks. Our results demonstrate that while IOTA exhibits high throughput and low latency in transaction processing, it sacrifices accuracy and becomes susceptible to orphanage attacks, particularly when the rate of incoming transactions is low. For example, when the rate of incoming transactions increases tenfold from  $0.3 \times 10^4$  to  $3 \times 10^4$  transactions per minute, IOTA's throughput increases by nearly tenfold, and the maximum latency decreases by almost 100%, but the accuracy declines by approximately 74%. In contrast, HybridChain maintains consistent accuracy, hovering around 99% for all incoming transaction rates, while experiencing a maximum latency of 10.25 seconds for a rate of  $3 \times 10^4$  transactions per minute. Additionally, Omniledger, as a sharded blockchain, achieves stable and acceptable accuracy by increasing the throughput through parallel processing in different shards. However, when the number of shards increases from 4 to 16, the maximum throughput decreases by 15% due to increased latency caused by cross-shard transactions.

The future direction includes delving deeper into the research potential of HybridChain, particularly in enhancing its architecture and algorithms. Future studies could explore the optimization of HybridChain's sharding mechanism and DAG structure to further improve scalability and security, especially in high-demand sectors like financial services or healthcare. Such research would contribute valuable insights into the adaptability and efficiency of HybridChain in handling diverse, real-world data loads and transaction types.

Additionally, the practical implementation and migration of existing Decentralized Applications (DApps) to HybridChain offers a broad scope for exploration. This includes assessing how HybridChain's unique features can be leveraged to enhance the performance and security of DApps, particularly in sectors that demand robust and scalable blockchain solutions. A focus on these implementations would provide critical feedback for ongoing improvements and adaptations in HybridChain's design.

Finally, addressing challenges like interoperability with different blockchain networks and ensuring user-friendly interfaces will be essential in establishing HybridChain as a versatile and

reliable platform for various applications. Pursuing these directions, grounded in practicality and innovation, sets the stage for HybridChain to make substantial contributions to decentralized system efficiencies and security.

## Chapter 4: SPID-Chain: A Smart Contract-Enabled, Polar-Coded Interoperable DAG Chain

### 4.1 Introduction

At present, the concept of Web3 is gaining momentum, heralding a new era of decentralized, interconnected, and verifiable digital ecosystems [121], [122]. Web3 envisions a fabric of interoperable blockchains, enabling seamless interactions and transactions across diverse networks. This vision extends beyond the mere exchange of data, encompassing the processing and validation of a myriad of transaction types in a distributed ledger ecosystem [123]–[125].

The evolution of blockchain technology has been marked by a shift from the initial focus on decentralization to a more nuanced emphasis on verifiability [126], [127]. While decentralization remains a cornerstone, the defining feature of Web3 has emerged as the ability to prove compliance with pre-agreed contractual terms within digital ecosystems, regardless of the underlying governance model or the complexity of the infrastructure [128], [129].

As we edge closer to this vision, the interoperability of blockchains becomes a crucial aspect of Web3, enabling a higher degree of efficiency and real-time responsiveness in various domains such as fintech [130], central banking [131], supply chain [132], and healthcare [133], [134]. For instance, consider a smart city framework where multiple blockchains orchestrate different facets of urban operations. Each blockchain, while optimizing operations within its domain, operates in isolation. However, a higher degree of efficiency and real-time responsiveness could be attained if these blockchains could communicate and reach consensus on cross-domain transactions [4], [135].

In the quest for blockchain interoperability, various approaches have emerged to address the challenges of connecting disparate blockchain networks. These approaches can be broadly categorized into five groups: *sidechains*, *notary schemes*, *hashed time lock contracts (HTLC)*, *relays*,

and *blockchain-agnostic protocols* [1]. Each group offers unique mechanisms and protocols to facilitate the transfer of assets, data, and transactions between different blockchain systems. Next, we discuss the related works concerning these interoperability approaches in detail.

#### 4.1.1 Related Works

To tackle the issue of blockchain interoperability, sidechains are employed. These are distinct blockchains linked to a primary blockchain, enabling asset exchanges in one or both directions. They enhance performance and facilitate asset utilization across different blockchain ecosystems. [136] proposes a sidechain method specifically for Proof-of-Stake (PoS) blockchains, introducing a cryptographic technique called ad-hoc threshold multi-signature (ATMS) for cross-chain certification, addressing security concerns such as the “goldfinger” attack in PoS blockchains. RootStock [137] is an example of a federated two-way peg sidechain for Bitcoin, which supports Ethereum-compatible smart contracts and uses a federation for asset management. Zendoo [8] offers a flexible protocol using zk-SNARKs for secure communication between the mainchain and sidechains.

Notary schemes involve trusted entities, known as notaries, to validate transactions across blockchains. They can be centralized or decentralized, with the latter involving a group of notaries. [4] introduces a decentralized notary scheme for secure cryptocurrency trading, reducing dependence on a single trusted entity. ReviewChain [138] uses smart contracts and notaries for verifying cross-chain review data. Notary Group [139] presents a model using a group-based notary scheme for asset exchange, increasing reliability and reducing the risk of failure. RenVM [140] combines Byzantine fault-tolerant (BFT) consensus with secure multi-party computation (MPC) for trustless asset transfers. Bifrost [141] employs a notary scheme for sharing data across different blockchains.

HTLC enable trustless atomic swaps of cryptocurrencies, ensuring that transactions are either completed or rejected for all parties involved. TierNolan [142] is credited with the original atomic swap protocol based on HTLCs. Herlihy [143] extends this to support atomic swaps in complex scenarios involving multiple parties and blockchains. Burn-to-Claim [144] uses HTLCs for secure

asset transfers between blockchains, involving burning assets on the source blockchain and recreating them on the destination blockchain.

Relays act as bridges for communication between different blockchain networks. BTCRelay [145] connects Bitcoin and Ethereum, allowing Ethereum users to verify Bitcoin transactions. XCLAIM [146] uses relays for trustless swaps between Bitcoin and Ethereum, employing a vault system for asset management. Verilay [147] is designed for Proof-of-Stake (PoS) blockchains, enabling validation of PoS protocols. Tesseract [148] uses Trusted Execution Environments (TEE) as trusted relays for secure cross-chain trades.

Blockchain-agnostic protocols provide a unified layer for interoperability across various blockchain networks. Interledger Protocol (ILP) v4 [149] supports secure and efficient asset transfers between different blockchains. Perun [150] enhances scalability by moving transactions off-chain into state channels. [151] propose a protocol for trusted data transfer across blockchain networks. Gravity [152] facilitates inter-blockchain communication and data exchange with an oracle consensus mechanism. Susy [152] builds on Gravity for cross-chain asset transfers, using oracle consensus for trust.

#### 4.1.2 Motivation and Contributions

The above schemes have various drawbacks, which motivate us to propose our smart contract-enabled Polar-coded interoperable DAG chain (SPID-Chain). In particular, sidechains and notary schemes, while pioneering in enabling blockchain interoperability, grapple with significant scalability and security challenges. Sidechains, for instance, rely heavily on the underlying security of the primary blockchain and often struggle to manage increased load, leading to bottlenecks [153]. Notary schemes, whether centralized or decentralized, introduce external validators which can become potential points of failure and centralization, thus compromising the decentralized ethos of blockchain technologies [154]. Our SPID-Chain's introduction of a DAG ledger addresses these scalability concerns by allowing parallel processing of transactions, significantly enhancing throughput without compromising security. Moreover, our approach mitigates centralization risks

associated with notary schemes by employing a bifurcated node architecture that ensures distributed validation without relying on a centralized group of validators [11].

HTLC and relay methods offer solutions for trustless asset swaps and communication between blockchains but often introduce high latency and operational complexity. These methods require locking assets into specific contracts with time-bound conditions, which can delay transaction completions and increase the complexity of multi-party transactions [155]. In contrast, our SPID-Chain utilizes an EDSC model that significantly reduces latency by responding to events rather than waiting for block confirmations. This model streamlines operations and enhances the user experience by facilitating quicker and more efficient interactions across blockchain networks [10].

Blockchain-agnostic protocols aim to create a unified layer for interoperability but often face challenges in maintaining efficiency and trust across diverse blockchain architectures. These protocols typically do not address the inherent differences in governance, consensus mechanisms, and transaction verification across chains, leading to inefficiencies and potential security vulnerabilities [156]. Our SPID-Chain's use of Polar codes for the polarization of computations presents a novel approach to addressing these inefficiencies. By optimizing the distribution of computational tasks, SPID-Chain enhances fault tolerance and overall network efficiency, ensuring robust cross-chain interactions that are both secure and scalable [157], [158].

We summarize the deficiencies of the existing methods and our SPID-Chain's solutions in Table.

4.1. The main contributions of this chapter are as follows:

1. We introduce a novel interoperability consensus for Web3, leveraging a bifurcated node architecture and a unique combination of a DAG ledger for inter-consensus and smart contracts for intra-consensus. This approach optimizes governance, validation processes, and computational workload distribution, enhancing both security and performance in a decentralized digital realm.
2. Our SPID-Chain features several innovative mechanisms, including the polarization of computations using Polar codes to address the straggler problem and improve fault tolerance, and an event-driven smart contract (EDSC) model that surpasses traditional transaction-driven

Table 4.1: Challenges and SPID-Chain solutions for blockchain interoperability.

Interoperability Approach	Challenges	SPID-Chain Solutions
Sidechains	<ul style="list-style-type: none"> <li>Scalability bottlenecks due to dependency on the main chain's security</li> <li>Security dependency on the main chain</li> </ul>	<ul style="list-style-type: none"> <li>DAG ledger allows parallel processing, improving scalability and throughput</li> <li>Independent security with bifurcated node architecture</li> </ul>
Notary Schemes	<ul style="list-style-type: none"> <li>Centralization risks</li> <li>Potential single points of failure</li> </ul>	<ul style="list-style-type: none"> <li>Distributed validation reduces centralization and enhances security</li> <li>Improves fault tolerance with innovative node architecture</li> </ul>
HTLC and Relays	<ul style="list-style-type: none"> <li>High latency</li> <li>Operational complexity in multi-party transactions</li> </ul>	<ul style="list-style-type: none"> <li>Event-driven smart contracts reduce latency by processing transactions based on events</li> <li>Simplifies cross-chain interactions, reducing dependency on sequential confirmations</li> </ul>
Blockchain-Agnostic Protocols	<ul style="list-style-type: none"> <li>Efficiency issues across diverse networks</li> <li>Trust and verification concerns</li> </ul>	<ul style="list-style-type: none"> <li>Utilizes Polar codes for efficient computational task distribution</li> <li>Enhances fault tolerance and security with distributed coded computing</li> </ul>

models in scalability, latency, and security.

- Through extensive simulations, our comprehensive scheme for cryptocurrency Web3 networks demonstrates excellent performances in terms of throughput, scalability, decentralization, security, and latency, showcasing its potential to facilitate seamless interactions and transactions across disparate blockchain networks.

The remainder of the chapter is organized as follows: Section 4.2 outlines the SPID-Chain architecture, discussing the consensus mechanisms and detailing cross-chain interactions via the DAG ledger. Section 4.3 examines the intra-consensus mechanism in SPID-Chain, focusing on event-driven smart contracts and their interactions with various network components. Section 4.4 presents the application of SPID-Chain in cryptocurrency Web3 networks. In Section 4.5, we

provide simulation results to evaluate the scheme’s performance on metrics such as throughput, scalability, decentralization, and security. Section 4.6 provides the conclusion of the chapter.

## 4.2 SPID-Chain

In this section, we first provide an overview of our proposed SPID-Chain and its consensus mechanisms. Then we describe cross-chain interactions through the DAG ledger.

### 4.2.1 System Overview

We consider a Web3 network consisting of  $N$  interconnected blockchains. We categorize nodes in each blockchain into two groups: committee nodes and worker nodes. Committee nodes are key players in the governance and validation processes within the chain, responsible for overseeing transactions and ensuring the integrity of the blockchain’s operations. Worker nodes, on the other hand, are tasked with executing computational work regarding processing the transactions. This bifurcation of roles allows for a more efficient and organized approach to managing blockchain operations, ensuring both security and performance.

SPID-Chain’s consensus mechanism comprises two key components: intra-consensus and inter-consensus. The intra-consensus component facilitates interactions among nodes within each blockchain, whereas the inter-consensus component is essential for achieving consensus across the blockchains, particularly for processing cross-chain transactions. Consequently, all interactions related to the intra-consensus of each chain are stored on its respective ledger. In contrast, data pertaining to cross-chain transactions and interactions among the blockchains are recorded on a DAG ledger. Therefore, while each blockchain maintains its individual ledger, the DAG ledger is collectively stored and managed by all blockchains.

Hence, within each chain committee nodes are cherry-picked from the ensemble of full nodes. Each full node possesses a complete copy of the blockchain’s ledger. Specifically, the ledger for each chain  $j$  maintained by full nodes is composed of two distinct parts. The first part includes data specific to chain  $j$ , such as events related to the activation of smart contracts or information

pertinent to committee selection (detailed in Section III). The second part comprises data common to all  $N$  chains, which includes the information contained in the blocks existing on the DAG. On the other hand, worker nodes, categorized as light nodes, retain only a fraction of the chain’s ledger in a coded format. Note that light nodes overwrite old data with new data, thereby sustaining a constant data storage requirement.

Table 4.2: Consensus mechanisms in SPID-Chain.

Consensus Type	Consensus Level	Mechanism/Tool	Purpose/Function
Inter-Consensus	Blockchain-Blockchain	DAG Ledger	Facilitates harmonious cross-chain transaction processing and data sharing across diverse blockchains, enhancing network interoperability.
Intra-Consensus	Committee-Committee Member	Smart Contracts	Oversees the decision-making process within committee nodes, crucial for the trustworthy creation of new blocks.
Intra-Consensus	Committee-Worker Member	Coded Distributed Computing	Enables efficient task execution and workload distribution, strengthening the cooperation between committee and worker nodes.

Our SPID-Chain, employs distinct mechanisms for managing both intra-consensus and inter-consensus processes, as outlined in Table 4.2. The DAG ledger is adeptly chosen for the blockchain-blockchain consensus due to its ability to handle a high volume of transactions efficiently. In contrast to traditional blockchains, where a linear sequence of blocks can slow down processing time, the DAG structure allows each node to add transactions independently without the need for synchronization across the entire network. This decentralized approach significantly enhances the scalability and speed of transaction processing. Moreover, the DAG ledger’s architecture is particularly suited for Web3 environments where multiple blockchains operate concurrently and asynchronously. By allowing each blockchain to contribute independently while maintaining a coherent global state, the DAG ledger ensures a smooth and efficient interoperability, which is critical for the robustness and reliability of a Web3 ecosystem.

Also, in the proposed interoperability consensus, smart contracts serve as a crucial component enhancing both automation and security. Their primary function is to automate the decision-making processes, ensuring a fair and secure method for orchestration of the tasks in the network.

Additionally, the integration of event logging within smart contracts is a significant advancement. This feature allows for the verification of function execution through logs, rather than requiring other committee nodes to re-execute the function. This approach is efficient in terms of resource utilization and time management. It also enhances security by providing a consistent and verifiable record of executions, thereby reducing the risk of discrepancies.

Moreover, coded distributed computing is strategically employed in the committee-worker member consensus to manage and optimize computational tasks. This is particularly vital in handling large-scale data inherent to Web3 environments. Coded distributed computing not only aids in mitigating the impact of stragglers and enhancing fault tolerance but also plays a pivotal role in reaching consensus. The latter is achieved by treating the computations performed by worker nodes as a form of “voting” for consensus. The final consensus value of the network is derived from these computational results, thereby ensuring that consensus is reached not only efficiently but also with the inherent benefits of fault tolerance and straggler mitigation.

#### 4.2.2 SPID-Chain Inter-consensus

In this subsection we first address inter-consensus interaction detailing how blockchains contribute to the DAG ledger in defined epochs. This is followed by discussing the consensus used for blocks on the DAG.

##### **Inter-Consensus Interaction**

Time is divided into epochs, and in each epoch  $t$ , every blockchain  $j$  has a primary objective: to generate and contribute a block, referred to as  $Z_j^P(t)$ , to the DAG ledger. This *proposed* block comprises transactions initiated within blockchain  $j$  and represents its contribution to that epoch.  $Z_j^P(t)$  then undergoes a consensus process (details in Section II-B-2) after being appended to the DAG ledger. If successful, it transitions from a proposed to a *confirmed* state, denoted as  $Z_j^C(t)$ , thereby finalizing the transactions it contains.

In each epoch  $t$ , blockchain  $j$  follows a specific process to generate its proposed block  $Z_j^P(t)$ .

Initially, it compiles all incoming transactions into a block, denoted as  $X_j^P(t)$ . To validate these transactions and formulate the proposed block, it references its cumulative history of confirmed blocks, denoted as  $\mathcal{Y}_j(t)$ , comprising all confirmed super-blocks  $\mathbf{Y}_j^C(t')$  added to blockchain  $j$ 's ledger from the genesis block up to the conclusion of the previous epoch  $t - 1$ , i.e.,  $1 \leq t' \leq t - 1$ . Each of these super-blocks  $\mathbf{Y}_j^C(t')$  contains a set of confirmed blocks  $Z_i^C(t'')$  on the DAG, i.e, the proposed block of chain  $i$  in epoch  $t''$  that is confirmed in epoch  $t' - 1$ . To create the corresponding block  $\mathbf{Y}_j^C(t')$  for each epoch  $t'$ , chain  $j$  selects one confirmed block from each of the  $N$  chains, provided that each chain has at least one confirmed block during epoch  $t'$ . If a chain has more than one confirmed block, chain  $j$  randomly chooses one of these blocks to include in  $\mathbf{Y}_j^C(t')$ . If a chain does not have any confirmed blocks, it is not included in  $\mathbf{Y}_j^C(t')$ . Thus by using  $\mathcal{Y}_j(t)$  as a reference for validation, chain  $j$  ensures that the transactions in the proposed block  $Z_j^P(t)$  are legitimate and align with the most recent blockchain history. Chain  $j$  then integrates the validated transactions in  $X_j^P(t)$  into  $Z_j^P(t)$  and submits it to the DAG ledger as its contribution to that epoch.

### DAG Ledger in Detail

By using a modified IOTA Coordicide [95] consensus mechanism as described next, our SPID-Chain employs a DAG ledger to facilitate cross-chain interactions and enhance interoperability in consensus mechanisms.

To submit the contributed block  $Z_j^P(t)$  for epoch  $t$  to the DAG, chain  $j$  generates a super-block  $\mathbf{Z}_j^T(t)$  comprising  $K$  randomly selected tip blocks for validation, resulting in  $\mathbf{Z}_j^V(t)$  the subset of valid blocks in  $\mathbf{Z}_j^T(t)$  using the information in the ledger  $\mathcal{Y}_j(t) = \bigcup_{t'=1}^{t-1} \mathbf{Y}_j^C(t')$ . It then integrates  $Z_j^P(t)$  into the DAG by selecting the blocks in  $\mathbf{Z}_j^V(t)$  as its predecessors, thereby establishing directed edges from the vertex of  $Z_j^P(t)$  to each block in  $\mathbf{Z}_j^V(t)$ . As a result of above actions performed by all chains, the aggregated weight (AW) of the blocks on the DAG changes. This leads to a change in the validity status of some blocks on the DAG. Specifically, certain proposed blocks, such as  $Z_i^P(t')$ , are confirmed, and their status are updated to  $Z_i^C(t')$ . Next, we explain how the AW of each block on the DAG is altered and how the transition from a proposed to a confirmed status occurs.

Fig. 5.1 illustrates a DAG ledger for two consecutive epochs  $t - 1$  and  $t$ . In epoch  $t - 1$ , each vertex represents a proposed block  $Z_{j_i}^P(t_i)$  by blockchain  $j_i$ , for  $i = 1, \dots, 4$ , at epoch  $t_i < t - 1$ . A direct edge from one vertex to another, for example, from  $Z_{j_2}^P(t_2)$  to  $Z_{j_1}^P(t_1)$ , signifies that  $j_2$  verifies block  $Z_{j_1}^P(t_1)$ . Fig. 5.1 also displays the status evolution of the blocks after one epoch. A block is labeled as a *tip* if it lacks incoming validation edges. Therefore, in this figure, for epochs  $t - 1$  and  $t$ , the blocks  $\{Z_{j_2}^P(t_2), Z_{j_3}^P(t_3), Z_{j_4}^P(t_4)\}$  and  $\{Z_{j_5}^P(t_5), Z_{j_6}^P(t_6)\}$  are tip blocks, respectively.

Moreover, a block that is neither a tip nor confirmed is referred to as an unconfirmed block. Hence, while blocks  $\{Z_{j_2}^P(t_2), Z_{j_3}^P(t_3), Z_{j_4}^P(t_4)\}$  are tips in epoch  $t - 1$ , block  $Z_{j_3}^P(t_3)$  is classified as unconfirmed in epoch  $t$ , while  $\{Z_{j_2}^P(t_2), Z_{j_4}^P(t_4)\}$  remain as tips. Observe that the status of  $Z_{j_1}^P(t_1)$ , transitions from proposed to confirmed, which is denoted by  $Z_{j_1}^C(t_1)$ . Specifically, a block achieves confirmed status when its AW surpasses a predefined threshold  $\eta$ . The AW is intricately linked to a weight vector that reflects the relative influence of the  $N$  blockchains.

The weight  $\omega_j$  of each blockchain  $j$  with  $\omega_j > 0$ ,  $\sum_{j=1}^N \omega_j = 1$  is determined by the number of tokens it has staked, and this weight is assumed to remain constant over different epochs. However, if blockchain  $j$  acts maliciously,  $\omega_j$  will be adjusted accordingly, as some of the staked tokens will be burnt. Essentially, the greater the number of tokens a blockchain stakes, the more significance the network assigns to its validations.

The AW of a block  $Z_{j_i}^P(t_i)$  is calculated as  $\omega_i + \sum_{Z_{j_{i'}}^P(t_{i'}) \in \mathcal{F}(Z_{j_i}^P(t_i))} \omega_{j_{i'}}$ , where  $\mathcal{F}(Z_{j_i}^P(t_i))$  denotes the future cone of  $Z_{j_i}^P(t_i)$ , encompassing all blocks that it validates, either directly or through a series of validations. For example, in Fig. 5.1 and in epoch  $t$ , blocks  $\{Z_{j_1}^P(t_1), Z_{j_4}^P(t_4)\}$  are in the future cone of block  $Z_{j_6}^P(t_6)$ , as block  $Z_{j_4}^P(t_4)$  is validated by it directly, while block  $Z_{j_6}^P(t_6)$  validates block  $Z_{j_1}^P(t_1)$  indirectly. Therefore, Fig. 5.1 illustrates the AW for the corresponding DAG ledgers, indicating, for example, how the AW of block  $Z_{j_1}^P(t_1)$  surpasses  $\eta$  and its status evolves from an unconfirmed block to confirmed after the addition of the weights of blocks  $Z_{j_5}^P(t_5)$  and  $Z_{j_6}^P(t_6)$  since block  $Z_{j_1}^P(t_1)$  is in the future cone of blocks  $Z_{j_5}^P(t_5)$  and  $Z_{j_6}^P(t_6)$ .

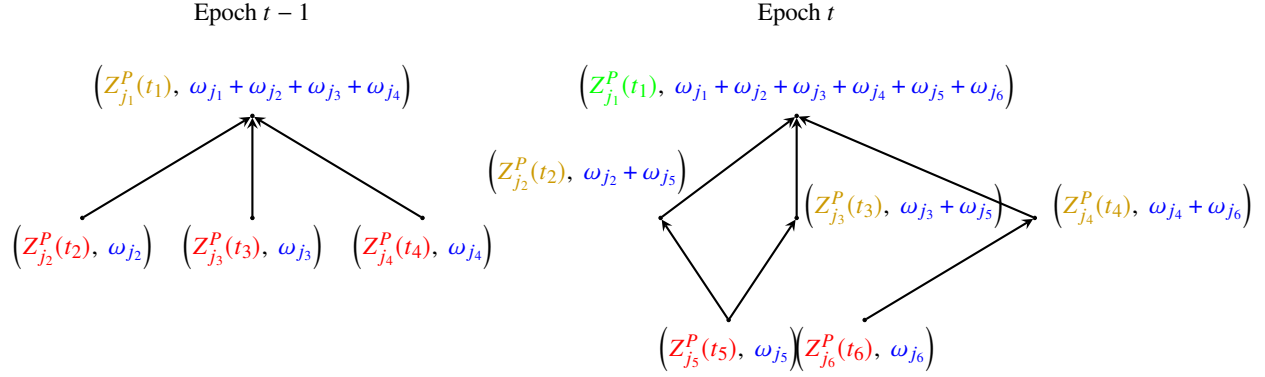


Figure 4.1: Evolution of a DAG ledger over two consecutive epochs. Nodes are color-coded to indicate their status: yellow for unconfirmed, green for confirmed, and red for tips. The blue text represents the AW associated with each node.

### 4.3 Intra-Consensus in SPID-Chain

In this section, we present the intra-consensus mechanism in SPID-Chain, focusing on the utilization of Event-Driven Smart Contracts (EDSC) and their interaction with oracles, committee nodes, and worker nodes. We discuss the role of smart contracts in various stages of consensus and outline the process for committee member selection and event validation to ensure network integrity and security.

#### 4.3.1 EDSC in SPID-Chain

##### Background on EDSC

Our SPID-Chain makes use of EDSC, which diverges from conventional smart contracts by leveraging three pivotal concepts: event definition, event subscription, and event publishing. These elements collectively enable a dynamic and responsive blockchain network.

Event definition involves delineating the characteristics of an event within the blockchain, such as changes in data values or the execution of specific actions. Once defined, these events are immutably recorded, ensuring security and permanence. Event subscription then follows, where smart contracts express interest in specific events. This phase involves integrating additional logic within the subscribing contracts, allowing for a tailored response when the event occurs. The

process culminates with event publishing, where the occurrence of an event is broadcast across the blockchain. This update triggers the subscribed smart contracts to autonomously execute predefined actions, based on their subscription logic.

These three stages form the backbone of the EDSC framework, enhancing the functionality and specificity of smart contracts within the blockchain ecosystem. The adoption of EDSC brings forth several advantages [159]. Firstly, it addresses scalability challenges by utilizing an event-driven model, allowing for more efficient processing and reduced network strain. This leads to improved performance and scalability. Secondly, the asynchronous nature of event handling in EDSC contributes to reduced latency in smart contract execution, enhancing system responsiveness. Additionally, EDSC's architecture inherently offers enhanced security features, mitigating the risk of certain attacks and ensuring that the execution of one contract does not directly impact others. Finally, the framework is well-suited for integrating external data through oracles, enabling more dynamic and responsive smart contract applications. Overall, EDSC represents a significant advancement in the realm of smart contracts, offering a more scalable, secure, and efficient framework for the blockchain ecosystem.

### **Oracle, Committee Nodes, and Worker Nodes**

As we discussed, during each epoch  $t$ , the goal of each blockchain  $j$  is to contribute a block  $Z_j^P(t)$  through an intra-consensus mechanism. This mechanism relies on EDSC, which orchestrates consensus stages using a set of six blocks  $\{C_j^k\}_{k=1}^6$  each containing a smart contract for specific intra-consensus tasks, and an oracle that publishes events to trigger these smart contracts. Moreover, in each epoch  $t$ , nodes in each blockchain  $j$  are split into committee members ( $C_j(t)$ ) and workers ( $W_j(t)$ ) for intra-consensus.

The oracle acts as a bridge between external data sources and the blockchain, generating events based on information provided by committee nodes or results from worker nodes. These events are then published to the network, where they await validation.

Committee nodes, responsible for maintaining the integrity and security of the network, validate

the events published by the oracle. They ensure that the events meet predefined criteria and are legitimate triggers for the smart contracts. Additionally, the highest-ranked committee node has the crucial role of feeding the oracle with information necessary for event generation, including transaction details, network conditions, or other relevant data.

Worker nodes come into play when a smart contract for processing is activated by a validated event. They are responsible for executing assigned tasks, such as processing data or performing computations. Upon completing their tasks, worker nodes submit the results to the oracle, potentially leading to the generation of new events.

#### 4.3.2 Smart Contracts and Consensus on Events

##### **Smart Contracts in SPID-Chain**

In our proposed scheme, each blockchain operates based on a sequence of stages in each epoch, with smart contracts playing a crucial role in ensuring compliance with these stages. We define smart contracts in six blocks, denoted as  $\{C_j^k\}_{k=1}^6$ , each designed to listen for specific events and activate at the appropriate times.

The events that trigger these smart contracts are predefined and can be published by various occurrences within the network. These occurrences include, but are not limited to, changes in data values and the execution of specific actions. The events are generated based on information provided by the network's nodes and are crucial for the realization of different stages within each epoch.

The first smart contract,  $C_j^1$ , is subscribed to the event published by the oracle pertaining to the formation of the transaction block  $X_j^P(t)$  initiated by accounts in chain  $j$  through committee nodes. This contract delegates the necessary computations to worker nodes for processing block  $X_j^P(t)$ . Consequently, the objective of this block is to assist committee nodes in first reaching consensus on  $X_j^P(t)$ , and second, on delineating the set of tasks to be assigned to worker nodes for processing  $X_j^P(t)$ .

The second smart contract,  $C_j^2$ , is subscribed to the event published by the oracle, which includes the computation results from worker nodes assigned by  $C_j^1$ . It generates  $Z_j^P(t)$ , the proposed block

in epoch  $t$  by chain  $j$ , contingent on the committee nodes' consensus on the validity of the published event. Thus, this contract aids in the consensus between worker and committee nodes for the creation of  $Z_j^P(t)$ . The computation results from the workers effectively act as votes on the validity of transactions in  $X_j^P(t)$ , leading to the selection of valid transactions and the generation of  $Z_j^P(t)$ . Additionally, as discussed later in Section III-B-2, the event that triggers  $C_j^2$  is verified by committee nodes and only activates  $C_j^2$  if there is consensus on its validity, further underscoring this contract's role in facilitating committee-worker consensus.

The third smart contract,  $C_j^3$ , is subscribed to the event published by the oracle regarding the formation of the super-block  $Z_j^T(t)$  for processing the tip blocks on the DAG by committee nodes. Analogous to  $C_j^1$ , the purpose of  $C_j^3$  is to aid committee nodes in reaching consensus on both  $Z_j^T(t)$  and the set of tasks to be assigned to worker nodes for processing  $Z_j^T(t)$ .

The fourth smart contract,  $C_j^4$ , is subscribed to the event published by the oracle concerning the reception of computation results from worker nodes assigned by  $C_j^3$ . The goal of this contract is to identify the super-block  $Z_j^V(t)$ , which consists of valid blocks in  $Z_j^T(t)$ . Similar to  $C_j^2$ , the purpose of  $C_j^4$  is to facilitate consensus between worker and committee nodes for the creation of super-block  $Z_j^V(t)$ .

The fifth smart contract,  $C_j^5$ , is subscribed to the event published by the oracle regarding the discovery and formation of  $Z_j^V(t)$  by committee nodes. The role of this contract is to submit the proposed block  $Z_j^P(t)$  to the DAG, update the AW of current blocks on the DAG, and, if eligible, change their status to confirmed. Hence,  $C_j^5$  assists committee nodes in reaching consensus on the *compliance* of submitting block  $Z_j^P(t)$  to the DAG as per the prescribed protocol.

The sixth and final smart contract,  $C_j^6$ , is subscribed to the event published by the oracle concerning the formation of the super-block  $Y_j^C(t)$ , which consists of confirmed blocks on the DAG by the committee nodes. The purpose of this contract is to update the ledger  $\mathcal{Y}_j(t)$ . Similar to  $C_j^5$ , this contract aids committee nodes in reaching consensus on the accuracy of  $Y_j^C(t)$ , the super-block added to the ledger in epoch  $t$ .

In summary, the smart contracts  $C_j^1$ ,  $C_j^3$ ,  $C_j^5$ , and  $C_j^6$  primarily facilitate consensus among

committee nodes, while  $C_j^2$  and  $C_j^4$  aim to enable consensus between committee and worker nodes.

### **Committee Member Selection and Event Validation**

The selection of committee members is based on a probability proportional to their stakes or reputations. Specifically, each node within a blockchain engenders a Verifiable Random Function (VRF) output utilizing their private key and a shared random seed, contingent on the current epoch time  $t$ . The VRF outputs are then used to determine the selection probability, with nodes boasting larger stakes or elevated reputations standing a higher chance of being elected as committee members. This mechanism not only earmarks the committee members but also ensures a fair and secure selection process vital for the network's integrity.

For smart contracts  $C_j^1$ ,  $C_j^3$ ,  $C_j^5$ , and  $C_j^6$ , committee nodes follow the following protocol to validate the published events by the oracle and handle the interactions with those smart contracts. The highest-ranked committee node feeds the relevant information to the oracle. The oracle publishes an event corresponding to the fed information. The published event acts as a proposed event, which is then validated by other committee members. This validation involves ensuring that the event proposal adheres to the agreed-upon criteria and rules, similar to how transaction validation works. The proposed event is subject to a vote by the committee members. If the majority approves the event, it becomes an active event. Now this active event triggers the corresponding smart contract that is subscribed to it. If the proposed event does not receive majority approval, it is discarded, and the next highest-ranked node gets the opportunity to propose an event by submitting the information to the oracle, leading the publication of a proposed event which in case of consensus on its validity will be an active event that will trigger the smart contract subscribed to it. This process continues until an event proposal gains the required majority consensus and executes its predefined logic based on the event data.

The protocol for activating the smart contracts  $C_j^2$  and  $C_j^4$  shares a similar structure with that of  $C_j^1$ ,  $C_j^3$ ,  $C_j^5$ , and  $C_j^6$ , with a focus on facilitating consensus between committee and worker nodes. In this process, the oracle publishes a proposed event containing computation results from worker

nodes. Committee nodes validate this event, and if it receives majority approval, it triggers the corresponding smart contract.

The approach of utilizing decentralized control in the execution of smart contracts offers several advantages. Firstly, it ensures that no single node possesses unilateral control over the triggering of smart contracts. This is because the activation of these contracts is subject to the approval of a committee, preventing any one entity from having excessive power. Additionally, this method significantly enhances security and integrity. The rank-based consensus required for event triggers means that only events that have been thoroughly vetted and approved can activate the smart contracts, thereby ensuring a higher level of trustworthiness and reliability in their execution. Furthermore, this approach retains the flexibility and efficiency inherent in EDSC. Smart contracts continue to be triggered by events, yet they benefit from an added layer of security through the decentralized consensus mechanism. This combination of decentralized control, enhanced security, and maintained efficiency makes this approach particularly effective in the management and execution of smart contracts in SPID-Chain.

## 4.4 SPID-Chain for Cryptocurrency Web3 Networks

### 4.4.1 Balance Checking

We consider a fundamental Web3 cryptocurrency framework designed to monitor balance transfers between accounts. We assume that each blockchain is linked to  $M$  accounts or addresses. The matrix  $\mathbf{X}_{\ell,\ell'}(t) \in \mathbb{R}^{M \times M}$  aggregates  $M$  transactions within epoch  $t$ . Specifically, a transaction within this block that involves account  $m$  on chain  $\ell$  transferring  $s$  tokens to account  $m'$  on chain  $\ell'$  is expressed as  $\mathbf{X}_{\ell,\ell'}(t)[m, m'] = s$ . Accounts that do not participate in sending tokens have their corresponding matrix entries set to zero.

To verify blocks for each chain  $j$  in each epoch  $t$ , represented as  $X_j^P(t) = \{\mathbf{X}_{j,\ell}(t)\}_{\ell=1}^N$ , we denote by  $P_j^t(X_j^P(t), \mathcal{Y}_j(t)) = Z_j^P(t)$  the operation that takes the formed block  $X_j^P(t)$  in epoch  $t$  and the data  $\mathcal{Y}_j(t)$  in the ledger updated up to epoch  $t$  and returns the proposed block  $Z_j^P(t) = \{\mathbf{Z}_{j,\ell}(t)\}_{\ell=1}^N$  of chain  $j$  for epoch  $t$  to attach to the DAG. To do so, for blocks  $\{\mathbf{X}_{j,\ell}(t)\}_{\ell=1}^N$  it is essential to

confirm that all sending accounts possess sufficient unspent funds from preceding transactions. Let  $\{\mathbf{Y}_{j,\ell}(r)\}_{\ell=1}^N$  represent the blocks confirmed from ledger  $\mathcal{Y}_j(t)$  that detail transactions spending tokens from accounts on chain  $j$  in epoch  $r$ . Similarly,  $\{\mathbf{Y}_{\ell,j}(r)\}_{\ell=1}^N$  refers to the blocks containing transactions that deposit tokens into accounts on chain  $j$  over the same epoch. We further define

$$\begin{aligned}\mathbf{A}_j(r) &= \sum_{\ell=1, \ell \neq j}^N \mathbf{Y}_{\ell,j}(r), \\ \mathbf{B}_j(r) &= \sum_{\ell=1, \ell \neq j}^N \mathbf{Y}_{j,\ell}(r), \quad r = 1, \dots, t, \\ \mathbf{C}_j(t) &= \sum_{\ell=1, \ell \neq j}^N \mathbf{X}_{j,\ell}(t).\end{aligned}\tag{4.1}$$

Here  $\mathbf{A}_j(r)$  captures the total tokens received by chain  $j$  from all other chains in epoch  $r$ ;  $\mathbf{B}_j(r)$  accounts for the total tokens dispatched from chain  $j$  to all other chains during the same epoch, and  $\mathbf{C}_j(r)$  denotes the tokens spent by chain  $j$  across all other chains in a given epoch  $t$ .

*Remark 1:* It is important to distinguish between  $\mathbf{X}_{j,\ell}(t)$ ,  $\mathbf{Z}_{j,\ell}(t)$ , and  $\mathbf{Y}_{j,\ell}(t)$  in terms of their roles within the transaction validation process. The block  $\mathbf{X}_{j,\ell}(t)$  represents initial transactions initiated by chain  $j$  targeting chain  $\ell$  within epoch  $t$ . Following intra-consensus validation, utilizing Eq.(4.3), this block becomes verified and is subsequently referred to as  $\mathbf{Z}_{j,\ell}(t)$ . Once  $\mathbf{Z}_{j,\ell}(t)$  is submitted to the DAG and successfully passes inter-consensus verification, it is recorded as  $\mathbf{Y}_{j,\ell}(t)$ .

*Remark 2:* Concerning the ledger  $\mathcal{Y}_j(t) = \bigcup_{t'=1}^{t-1} \mathbf{Y}_j^C(t')$  as introduced in Section II-B-2, note that this collection comprises not only the confirmed transaction blocks  $\{\mathbf{Y}_{j,\ell}(t')\}_{\ell=1}^N$  from chain  $j$  but also includes analogous blocks from other chains, specifically  $\{\mathbf{Y}_{j',\ell}(t')\}_{\ell=1}^N$  for any  $j' \neq j$ . This aggregation ensures a comprehensive record of cross-chain transactions, facilitating a more robust and secure verification process across the network.

Next we define the cumulative matrices  $\mathbf{W}_j^{in}(t)$  and  $\mathbf{W}_j^{out}(t)$  as:

$$\mathbf{W}_j^{in}(t) = \sum_{r=1}^t \mathbf{A}_j(r), \quad \mathbf{W}_j^{out}(t) = \sum_{r=1}^t \mathbf{B}_j(r) + \mathbf{C}_j(t),\tag{4.2}$$

where  $\mathbf{W}_j^{in}(t)$  represents the total inflow of tokens to chain  $j$  from other chains through epoch  $t$ , while  $\mathbf{W}_j^{out}(t)$  summarizes the total outflow from chain  $j$  to other chains by epoch  $t$ . It should be noted that for  $r = t$ ,  $\mathbf{B}_j(t)$  represents the confirmed spending transactions for epoch  $t$ , whereas  $\mathbf{C}_j(t)$  does not include confirmed transactions. Instead,  $\mathbf{C}_j(t)$  consists of proposed transactions that accounts on chain  $j$  intend to execute during epoch  $t$ .

A transaction involving account  $m$  in chain  $j$  during epoch  $t$  is deemed valid if the following condition is satisfied:

$$\begin{aligned} \mathbf{w}_j(t)[m] &= \sum_{m'=1}^M \mathbf{W}_j^{in}(t)[m', m] \\ &\quad - \sum_{m'=1}^M \mathbf{W}_j^{out}(t)[m, m'] \geq 0, \quad m = 1, \dots, M, \end{aligned} \quad (4.3)$$

where,  $\mathbf{w}_j(t)[m]$  represents the net balance of account  $m$  at epoch  $t$ .

Then to compute the valid blocks  $\mathbf{Z}_{j,\ell}(t)$  from each transaction block  $\mathbf{X}_{j,\ell}(t)$ , and consequently to obtain  $Z_j^P(t) = \{\mathbf{Z}_{j,\ell}(t)\}_{\ell=1}^N$ , for each account  $m$  in chain  $j$  that  $\mathbf{w}_j(t)[m] \geq 0$  we set  $\mathbf{Z}_{j,\ell}(t)[m, m'] = \mathbf{X}_{j,\ell}(t)[m, m']$ , and otherwise it is set to zero.

*Remark 3:* In SPID-Chain, in addition to  $X_j^P(t)$ , each chain  $j$  validates a subset of tip blocks during each epoch  $t$ . Specifically, it processes a set of  $K$  tip blocks,  $\mathbf{Z}_j^T(t)$ , using the operation  $V_j^t(\mathbf{Z}_j^T(t), \mathcal{Y}_j(t)) = \mathbf{Z}_j^V(t)$  to produce the validated blocks  $\mathbf{Z}_j^V(t)$ . Each super-block  $\mathbf{Z}_j^T(t)$  includes blocks  $Z_i^P(t_i)$  proposed by different chains  $i \neq j$  at epoch  $t_i \leq t$ , with a restriction of including no more than one block from each chain.

For each block  $Z_i^P(t_i) = \{\mathbf{Z}_{i,\ell}(t_i)\}_{\ell=1}^N$  within  $\mathbf{Z}_j^T(t)$ , corresponding to (4.1) we denote:

$$\begin{aligned} \mathbf{A}_i(r) &= \sum_{\ell=1, \ell \neq i}^N \mathbf{Y}_{\ell,i}(r), \\ \mathbf{B}_i(r) &= \sum_{\ell=1, \ell \neq i}^N \mathbf{Y}_{i,\ell}(r), \quad r = 1, \dots, t, \\ \mathbf{C}_i(t) &= \sum_{\ell=1, \ell \neq i}^N \mathbf{Z}_{i,\ell}(t_i). \end{aligned} \quad (4.4)$$

Then chain  $j$  validates the  $K$  blocks  $Z_i^P(t_i)$  in  $\mathbf{Z}_j^T(t)$  to obtain  $\mathbf{Z}_j^V(t)$  by following the similar process for validating  $X_j^P(t) = \{\mathbf{X}_{j,\ell}(t)\}_{\ell=1}^N$  as described by (4.2) and (4.3).

#### 4.4.2 Coded Verification

##### Polar Encoding

We assume each chain  $j$  has  $n$  workers, and the probabilities  $\{p_k\}_{k=1}^n$  of worker  $k$  being a straggler are given. Define  $\lambda = \frac{1}{n} \sum_{k=1}^n p_k$ . Then the rate of the Polar code is  $R = 1 - \lambda$ . Moreover, to describe the encoding process, we first form the set of stragglers  $\mathcal{S}$  consisting of worker nodes corresponding to the  $|\mathcal{S}| = \lambda n$  highest values among  $\{p_k\}_{k=1}^n$ .

First assume that  $n$  is power of 2. To encode the  $M \times M$  matrix  $\mathbf{A}_j(r)$  in (4.1), its  $M$  rows are first evenly partitioned into  $n(1 - \lambda)$  sets, and each resulting  $\frac{M}{n(1-\lambda)} \times M$  submatrix is assigned to a non-straggler worker. And for a straggler worker in  $\mathcal{S}$ , a  $\frac{M}{n(1-\lambda)} \times M$  all-zero matrix is assigned. This way an expanded matrix

$$\hat{\mathbf{A}}_j(r) = \begin{bmatrix} \mathbf{a}_{j,1}^\top(r) & \cdots & \mathbf{a}_{j,k}^\top(r) & \cdots & \mathbf{a}_{j,n}^\top(r) \end{bmatrix}^\top \in \mathbb{R}^{\frac{M}{1-\lambda} \times M} \quad (4.5)$$

is obtained where  $\mathbf{a}_{j,k}^\top(r)$  has dimension  $\frac{M}{n(1-\lambda)} \times M$ . If  $k \in \mathcal{S}$  then it is an all-zero matrix, otherwise it is a submatrix of  $\mathbf{A}_j(r)$ .

Next an  $n \times n$  Hadamard transform is applied to the  $n \times 1$  block matrix in (4.5), resulting in the Polar coded matrix

$$\begin{aligned} \mathbb{H}_n\{\hat{\mathbf{A}}_j(r)\} &= \left[ \tilde{\mathbf{a}}_{j,1}^\top(r), \dots, \tilde{\mathbf{a}}_{j,k}^\top(r), \dots, \tilde{\mathbf{a}}_{j,n}^\top(r) \right]^\top \\ &= \tilde{\mathbf{A}}_j(r) \in \mathbb{R}^{\frac{M}{1-\lambda} \times M}. \end{aligned} \quad (4.6)$$

where each  $\tilde{\mathbf{a}}_{j,k}^\top(r)$  is a  $\frac{M}{n(1-\lambda)} \times M$  non-zero matrix.

Next consider the case that  $n$  is not a power of 2. Then we can write  $n = n_L + n_{L-1} + \dots + n_1$ , where  $n_L > n_{L-1} > \dots > n_1$  and each  $n_k$  is a power of 2. For example, if  $n = 100$ , then  $L = 3$ , and  $n_3 = 64$ ,  $n_2 = 32$ , and  $n_1 = 4$ .

Let  $M_\kappa = \frac{n_\kappa}{n}M$ ,  $\kappa = 1, \dots, L$ . Then  $M = M_L + \dots + M_1$ . We divide the  $n$  worker nodes into  $L$  groups, with the  $\kappa$ -th group consist of  $n_\kappa$  worker nodes, and the corresponding straggler set  $\mathcal{S}_\kappa$  with  $|\mathcal{S}_\kappa| = n_\kappa\lambda$ .

The  $\kappa$ -th group encodes  $M_\kappa$  rows of the  $M \times M$  matrix  $\mathbf{A}_j(r)$  by first forming the corresponding expanded matrix  $\hat{\mathbf{A}}_j^\kappa(r) \in \mathbb{R}^{\frac{M_\kappa}{1-\lambda} \times M}$  similar to (4.5), with  $n_\kappa$  row-blocks, and then applying an  $n_\kappa \times n_\kappa$  Hadamard transform, resulting in Polar coded matrix  $\tilde{\mathbf{A}}_j^\kappa(r) \in \mathbb{R}^{\frac{M_\kappa}{1-\lambda} \times M}$  similar to (4.6) with each row-block  $\tilde{\mathbf{a}}_{j,k,\kappa}^\top(r)$  of dimension  $\frac{M_\kappa}{n_\kappa(1-\lambda)} \times M$ .

## Coded Computing

Note that the uncoded computation in (4.2) can be written recursively as

$$\mathbf{W}_j^{in}(t) = \mathbf{W}_j^{in}(t-1) + \mathbf{A}_j(t), \quad (4.7a)$$

$$\mathbf{W}_j^{out}(t) = \mathbf{W}_j^{out}(t-1) + \mathbf{B}_j(t) + \Delta\mathbf{C}_j(t), \quad (4.7b)$$

where  $\Delta\mathbf{C}_j(t) = \mathbf{C}_j(t) - \mathbf{C}_j(t-1)$ . Correspondingly the coded versions of these updates are given by:

$$\tilde{\mathbf{W}}_j^{in}(t) = \tilde{\mathbf{W}}_j^{in}(t-1) + \tilde{\mathbf{A}}_j(t), \quad (4.8a)$$

$$\tilde{\mathbf{W}}_j^{out}(t) = \tilde{\mathbf{W}}_j^{out}(t-1) + \tilde{\mathbf{B}}_j(t) + \Delta\tilde{\mathbf{C}}_j(t). \quad (4.8b)$$

In the SPID-Chain, committee nodes, i.e., full nodes, are responsible for storing blocks in raw format. Worker nodes, on the other hand, manage coded data. In particular, assuming that  $n$  is a power of 2, each worker node  $k$  stores coded matrices:

$$\tilde{\mathbf{w}}_{j,k}^{in}(t) = \sum_{r=1}^t \tilde{\mathbf{a}}_{j,k}^\top(r), \quad \tilde{\mathbf{w}}_{j,k}^{out}(t) = \sum_{r=1}^t \tilde{\mathbf{b}}_{j,k}^\top(r) + \Delta\tilde{\mathbf{c}}_{j,k}^\top(t), \quad (4.9)$$

which correspond to the  $k$ -th row-block of the coded matrices in (4.8).

Then to compute (4.8a) and (4.8b) at each epoch  $t$ , SPID-Chain commences with full nodes

encoding the current matrices  $\mathbf{A}_j(t)$ ,  $\mathbf{B}_j(t)$ , and  $\Delta\mathbf{C}_j(t)$  to obtain the encoded matrices  $\tilde{\mathbf{A}}_j(t)$ ,  $\tilde{\mathbf{B}}_j(t)$ , and  $\Delta\tilde{\mathbf{C}}_j(t)$  according to (4.6). Then with the help of some smart contracts committee nodes distribute the data  $\tilde{\mathbf{a}}_{j,k}^\top(t)$ ,  $\tilde{\mathbf{b}}_{j,k}^\top(t)$ , and  $\Delta\tilde{\mathbf{c}}_{j,k}^\top(t)$  to designated worker nodes. Upon receipt, each worker node updates its stored coded matrices by incorporating the new encoded data:

$$\tilde{\mathbf{w}}_{j,k}^{in}(t) = \tilde{\mathbf{w}}_{j,k}^{in}(t-1) + \tilde{\mathbf{a}}_{j,k}^\top(t), \quad (4.10a)$$

$$\tilde{\mathbf{w}}_{j,k}^{out}(t) = \tilde{\mathbf{w}}_{j,k}^{out}(t-1) + \tilde{\mathbf{b}}_{j,k}^\top(t) + \Delta\tilde{\mathbf{c}}_{j,k}^\top(t). \quad (4.10b)$$

Finally, worker nodes return the updated  $\tilde{\mathbf{w}}_{j,k}^{in}(t)$  and  $\tilde{\mathbf{w}}_{j,k}^{out}(t)$  to the committee nodes for decoding and obtaining (4.8a) and (4.8b), respectively, facilitated by a smart contract and an oracle. Specifically, committee nodes assess whether the results received from worker nodes are decodable [160], i.e., if  $\mathbf{W}_j^{in}(t)$  and  $\mathbf{W}_j^{out}(t)$  can be reconstructed based on the received results from the worker nodes. Once confirmed as decodable, the committee nodes implement a recursive decoding algorithm [160], [161]. For details on the recursive decoding algorithm see Algorithms 1 and 2 in [160].

*Remark 4:* When  $n$  is not a power of 2, coded computing is carried out independently in  $L$  groups. In particular, within each group  $\kappa$ , the expressions in (4.7) now involve  $\mathbf{W}_{j,\kappa}^{in}(t)$ ,  $\mathbf{W}_{j,\kappa}^{out}(t)$ ,  $\mathbf{A}_j^\kappa(t)$ ,  $\mathbf{B}_j^\kappa(t)$ ,  $\Delta\mathbf{C}_j^\kappa(t)$ , while those in (4.8) involve  $\tilde{\mathbf{W}}_{j,\kappa}^{in}(t)$ ,  $\tilde{\mathbf{W}}_{j,\kappa}^{out}(t)$ ,  $\tilde{\mathbf{A}}_j^\kappa(t)$ ,  $\tilde{\mathbf{B}}_j^\kappa(t)$ , and  $\Delta\tilde{\mathbf{C}}_j^\kappa(t)$ . Then, each worker node  $k$  in group  $\kappa$  stores coded matrices corresponding to (4.9), denoted as  $\tilde{\mathbf{w}}_{j,k,\kappa}^{in}(t)$  and  $\tilde{\mathbf{w}}_{j,k,\kappa}^{out}(t)$  which the procedure for updating these matrices are similar to (4.10) and are formed by the row-blocks  $\tilde{\mathbf{a}}_{j,k,\kappa}^\top(t)$ ,  $\tilde{\mathbf{b}}_{j,k,\kappa}^\top(t)$ , and  $\Delta\tilde{\mathbf{c}}_{j,k,\kappa}^\top(t)$ , i.e., the  $k$ -th row-block of the encoded matrices  $\tilde{\mathbf{A}}_j^\kappa(t)$ ,  $\tilde{\mathbf{B}}_j^\kappa(t)$ , and  $\Delta\tilde{\mathbf{C}}_j^\kappa(t)$ , respectively.

Moreover, the decoding process is also performed independently, in each group. Specifically, each worker node  $k$  in group  $\kappa$  sends back the intermediate values  $\tilde{\mathbf{w}}_{j,k,\kappa}^{in}(t)$  and  $\tilde{\mathbf{w}}_{j,k,\kappa}^{out}(t)$ . Then, committee nodes decode these results to obtain the matrices  $\mathbf{W}_{j,\kappa}^{in}(t)$  and  $\mathbf{W}_{j,\kappa}^{out}(t)$ . Once results are obtained for all  $L$  groups, the final  $\mathbf{W}_j^{in}(t)$  and  $\mathbf{W}_j^{out}(t)$  are formed.

*Remark 5-motivation for coded computing:* In case of uncoded computation each worker  $k$  stores an  $\frac{1}{n}$  portion of matrices  $\mathbf{A}_j(t)$ ,  $\mathbf{B}_j(t)$ , and  $\Delta\mathbf{C}_j(t)$  as  $\mathbf{a}_{j,k}^\top(t)$ ,  $\mathbf{b}_{j,k}^\top(t)$ , and  $\Delta\mathbf{a}_{j,k}^\top(t)$ , respectively,

and computes  $\mathbf{w}_{j,k}^{in}(t) = \sum_{r=1}^t \mathbf{a}_{j,k}^\top(r)$  and  $\mathbf{w}_{j,k}^{out}(t) = \sum_{r=1}^t \mathbf{b}_{j,k}^\top(r) + \Delta \mathbf{c}_{j,k}^\top(t)$ . Then during block verification, to obtain  $\mathbf{W}_j^{in}(t)$  and  $\mathbf{W}_j^{out}(t)$ , as the system must wait for results from all workers, and therefore the slowest worker is the bottleneck. On the other hand, with coded computation,  $\tilde{\mathbf{w}}_{j,k}^{in}(t)$  and  $\tilde{\mathbf{w}}_{j,k}^{out}(t)$  are linear combinations of  $\mathbf{a}_{j,k}^\top(t)$ ,  $\mathbf{b}_{j,k}^\top(t)$ , and  $\Delta \mathbf{c}_{j,k}^\top(t)$  and such inherent redundancy ensures timely completion of the computation  $\mathbf{W}_j^{in}(t)$  and  $\mathbf{W}_j^{out}(t)$ , even in the presence of slowest workers. Once the fastest workers complete their tasks, they send results back to the committee nodes, which then decode them without the need to wait for the slow workers.

*Remark 6-motivation of Polar codes:* Polar codes maintain a consistent error rate even as block lengths expand, ensuring efficient handling of both large and small computational tasks [162]. This consistent performance is crucial in Web3 scenarios, where tasks frequently involve managing varying amounts of data across multiple nodes, providing needed reliability and stability. Additionally, the encoding and decoding processes of Polar codes are less complex compared to other well-known codes like Reed-Muller or random linear codes [162]. This is particularly advantageous in Web3 networks, where quick data processing enhances system efficiency and resource optimization. Next we detail the structure of the smart contracts for token transferring.

#### 4.4.3 Structure of the Smart Contracts for Token Transferring

For verifying token transfer transactions, we delineate the six blocks  $\{C_j^k\}_{k=1}^6$  maintained by the committee nodes and their respective roles in each epoch  $t$ . The input, output, and state of each smart contract are denoted by  $\{I_j^k(t), O_j^k(t), S_j^k(t)\}_{k=1}^6$ , respectively.

#### Encoding and Decoding Processing Blocks

The initial two blocks,  $C_j^1$  and  $C_j^2$ , are allocated for encoding and decoding the *processing* block, respectively. To compute  $P_j^t(X_j^P(t), \mathcal{Y}_j(t)) = Z_j^P(t)$ , the smart contracts in these blocks are associated with the *defined major events* of forming  $X_j^P(t)$  and computing the results of the worker to ascertain  $Z_j^P(t)$ , respectively.

**Encoding Processing Block**  $C_j^1$  is *subscribed* to the event *published* by the oracle resulting from the formation of block  $X_j^P(t)$  by the committee nodes. Specifically, matrices  $\mathbf{A}_j(t)$ ,  $\mathbf{B}_j(t)$ , and  $\Delta\mathbf{C}_j(t)$  from (4.1) are formed by the committee nodes and provided to the oracle along with additional information such as the number of the worker nodes  $n$  and their public keys. Upon reception of such information, the oracle publishes an event, and upon its validation by committee nodes, it triggers  $C_j^1$ .

- $I_j^1(t)$ : Upon triggering, the input for  $C_j^1$  serves as the encoder for the Polar code discussed in Section IV-B. This encoder initially forms the corresponding matrices  $\hat{\mathbf{A}}_j(t)$ ,  $\hat{\mathbf{B}}_j(t)$ , and  $\Delta\hat{\mathbf{C}}_j(t) = \hat{\mathbf{C}}_j(t) - \hat{\mathbf{C}}_j(t-1)$  using the information in the event published by the oracle as the result of the data provided by the committee nodes and then generates encoded matrices  $\tilde{\mathbf{A}}_j(t)$ ,  $\tilde{\mathbf{B}}_j(t)$ , and  $\Delta\tilde{\mathbf{C}}_j(t)$ .
- $O_j^1(t)$ : The output for  $C_j^1$  comprises the set of computation tasks  $\tilde{\mathbf{a}}_{j,k}^\top(t)$ ,  $\tilde{\mathbf{b}}_{j,k}^\top(t)$ , and  $\Delta\tilde{\mathbf{c}}_{j,k}^\top(t)$  that are dispatched to each worker  $k$ .
- $S_j^1(t)$ : After assigning the computation tasks to workers, the state of  $C_j^1$  transitions to complete, and it utilizes an external entity to dispatch the assigned tasks to the workers.

**Decoding Processing Block**  $C_j^2$  is *subscribed* to the event *published* by the oracle as a consequence of receiving sufficient computation tasks from worker nodes. Thus, this smart contract corresponds to the *defined major event* of acquiring adequate computation results from worker nodes. Specifically, as delineated in Section IV-B, upon receiving enough decodable results from worker nodes, the oracle publishes an event and forwards this event to the committee nodes for verification and achieving consensus on its validity. Upon validation of the published event, it triggers  $C_j^2$ . Each worker node  $k$  initially updates the stored encoded matrices  $\tilde{\mathbf{w}}_{j,k}^{in}(t)$  and  $\tilde{\mathbf{w}}_{j,k}^{out}(t)$  from (4.10) and transmits them back to the oracle in order to obtain (4.8a) and (4.8b), respectively.

- $I_j^2(t)$ : The input for this smart contract leverages the computation results from workers and, firstly, utilizing the decoder embedded in  $C_j^2$  obtains  $\mathbf{W}_j^{in}(t)$  and  $\mathbf{W}_j^{out}(t)$  in (4.7) and then verifies the transactions in  $X_j^P(t)$  from (4.3).

- $O_j^2(t)$ : With the verified transactions in  $X_j^P(t)$ , the output for  $C_j^2$  is  $Z_j^P(t)$ , the proposed block for epoch  $t$ .
- $S_j^2(t)$ : Upon successfully determining  $Z_j^P(t)$ , the state of  $C_j^2$  transitions to complete.

### Encoding and Decoding Validation Blocks

Blocks  $C_j^3$  and  $C_j^4$  in the SPID-Chain function similarly to the initial encoding and decoding blocks,  $C_j^1$  and  $C_j^2$ , but are specifically designed to handle multiple tip blocks on the DAG.

**Encoding Validation Block**  $C_j^3$ , the encoding validation block, parallels the operations of  $C_j^1$  but is triggered for each  $Z_i^P(t_i)$  in  $\mathbf{Z}_j^T(t)$ , for  $t_i \leq t$  at the tip of the DAG, repeating the process for each block, up to a total of  $K$  blocks.  $C_j^3$  processes input similarly to  $C_j^1$ , but it applies the encoding operations to multiple blocks from different chains, each represented once among the tip blocks to obtain  $\tilde{\mathbf{A}}_i(t)$ ,  $\tilde{\mathbf{B}}_i(t)$  and  $\Delta\tilde{\mathbf{C}}_i(t)$ , refer to (4.4) for each block  $Z_i^P(t_i)$  in  $\mathbf{Z}_j^T(t)$ .

**Decoding Validation Block** Similarly,  $C_j^4$ , the decoding validation block, mirrors  $C_j^2$ 's functionality but extends it to handle multiple blocks at the DAG's tip. It is triggered upon receiving result of the computation tasks  $\tilde{\mathbf{w}}_{i,k}^{in}(t)$  and  $\tilde{\mathbf{w}}_{i,k}^{out}(t)$  for each of these blocks, ensuring that the decoding and validation processes are executed repetitively for up to  $K$  blocks.

Upon successfully determining valid transaction in  $Z_i^P(t_i)$  for all blocks in  $\mathbf{Z}_j^T(t)$ , block  $\mathbf{Z}_j^V(t)$  is determined.

### Submission Block

Block  $C_j^5$  is designated as the *submission* block. The smart contract within this block is utilized in SPID-Chain to submit the block  $Z_j^P(t)$  to the DAG. As discussed in Section III-B-1, the smart contract in this block corresponds to the *defined major event* of forming  $\mathbf{Z}_j^V(t)$ . Specifically,  $C_j^5$  is *subscribed* to the event *published* by the oracle resulting from the formation of block  $\mathbf{Z}_j^V(t)$  by the committee nodes. When block  $\mathbf{Z}_j^V(t)$  is formulated by the committee nodes and provided to the oracle, it publishes an event that, upon validation, triggers  $C_j^5$ .

- $I_j^5(t)$ : Upon triggering, the input  $I_j^5(t)$  for  $C_j^5$  identifies the valid blocks within  $\mathbf{Z}_j^V(t)$  on the DAG. Considering that the vertices corresponding to blocks in  $\mathbf{Z}_j^V(t)$  serve as the parents for  $\mathbf{Z}_j^P(t)$ , the function embedded in  $C_j^5$  updates the AW of the vertices on the DAG.
- $O_j^5(t)$ : Consequently, the output of  $C_j^5$  alters the status of those blocks in the DAG whose AW exceeds a predetermined threshold from *proposed* to *confirmed*.
- $S_j^5(t)$ : After successfully obtaining the output of  $C_j^5$ , the state of  $C_j^5$  releases the proposed block  $\mathbf{Z}_j^P(t)$  for epoch  $t$  into the DAG by creating a new vertex. This vertex establishes directed edges to vertices representing the parent blocks in  $\mathbf{Z}_j^V(t)$ , which contain the valid blocks.

### Updating Block

$C_j^6$  is designated as the *updating* block. The smart contract within this block is utilized in SPID-Chain to form  $\mathbf{Y}_j^C(t+1)$  and consequently update the ledger to ascertain  $\mathcal{Y}_j(t+1)$ . As discussed in Section III-B-1, the smart contract in this block corresponds to the *defined major event* of forming super-block  $\mathbf{Y}_j^C(t+1)$ . Specifically,  $C_j^6$  is *subscribed* to the event *published* by the oracle resulting from the formation of block  $\mathbf{Y}_j^C(t+1)$  by the committee nodes. To form  $\mathbf{Y}_j^C(t+1)$  committee nodes in chain  $j$  take only one block from those chains that have at least one block. Then, upon providing the block  $\mathbf{Y}_j^C(t+1)$  to the oracle, it publishes an event that, upon validation, triggers  $C_j^6$ .

- $I_j^6(t)$ : Upon triggering, the input for  $C_j^6$  incorporates the transaction into  $\mathcal{Y}_j(t+1)$  using the information in the event, which essentially comprises the confirmed transactions in  $\mathbf{Y}_j^C(t+1)$ .
- $O_j^6(t)$ : As a consequence of the input, the output of  $C_j^6$  yields the ledger  $\mathcal{Y}_j(t+1)$ , which is maintained by the committee nodes and is utilized for the subsequent epoch.
- $S_j^6(t)$ : After successfully determining  $\mathcal{Y}_j(t+1)$ , the state of  $C_j^6$  transitions to complete.

Table. 5.1 summarizes the structure of the smart contracts for token transferring.

Table 4.3: Summary of Smart Contracts for Token Transferring

Block	Major Defined Event Subscription	Input	Output	State
$C_j^1$	Formation of $X_j^P(t)$	Encoder using matrices $\hat{\mathbf{A}}_j(t)$ , $\hat{\mathbf{B}}_j(t)$ , $\Delta\hat{\mathbf{C}}_j(t)$	Set of computation tasks $\tilde{\mathbf{a}}_{j,k}^\top(t)$ , $\tilde{\mathbf{b}}_{j,k}^\top(t)$ , $\Delta\tilde{\mathbf{c}}_{j,k}^\top(t)$	Complete after dispatching tasks to the workers.
$C_j^2$	Receiving enough computation results	Leveraging computation results from workers to decode and verify transactions	Proposed block for epoch $t$ , $Z_j^P(t)$	Complete after finding $Z_j^P(t)$ .
$C_j^3$	Formation of $\mathbf{Z}_j^T(t)$	Encoder using matrices $\hat{\mathbf{A}}_i(t)$ , $\hat{\mathbf{B}}_i(t)$ , $\Delta\hat{\mathbf{C}}_i(t)$	Set of computation tasks for worker nodes	Complete after dispatching tasks to the workers.
$C_j^4$	Receiving enough computation results	Utilizing computation results from workers to decode and verify blocks	Validated transactions in $Z_i^P(t_i)$	Complete after finding $\mathbf{Z}_j^V(t)$ .
$C_j^5$	Formation of $\mathbf{Z}_j^V(t)$	Identifying valid blocks in $\mathbf{Z}_j^V(t)$ on the DAG	Updating AW and changing status of blocks in DAG from <i>proposed</i> to <i>confirmed</i>	Releases the proposed block $Z_j^P(t)$ into the DAG.
$C_j^6$	Formation of $\mathbf{Y}_j^C(t+1)$	Adding transactions to $\mathcal{Y}_j(t+1)$	Ledger $\mathcal{Y}_j(t+1)$	Complete after updating the ledger.

#### 4.4.4 Stages of SPID-Chain

For each set of six smart contracts,  $\{C_j^i\}_{i=1}^6$ , triggered by a confirmed even, i.e., an event that has been validated through consensus by the committee nodes—we denote the confirmed event as  $\mathcal{X}_j^i(t)$ . This event triggers the corresponding smart contracts  $\{C_j^i\}_{i=1}^6$  in epoch  $t$ . Furthermore, we denote by  $\mathcal{T}_j$  a temporary pool consisting of events  $\mathcal{X}_j^i(t)$ . This pool is populated at the start of each epoch  $t$  and is emptied at the epoch’s end, following the completion of all stages in the SPID-Chain’s protocol, and its contents are recorded in a block denoted by  $\mathcal{D}_j(t)$  which is the  $t$ -th block in the side event ledger  $\mathcal{D}_j$ . Therefore, the operation of SPID-Chain network is driven by a sequence of stages, executed by each chain  $j$  in every epoch  $t$ . These stages are:

### **Stage 1: Proposed Block Generation:**

*Step 1: Computation tasks for computing  $P_j^t$  are assigned to the workers:* Following committee-committee consensus block  $X_j^P(t)$  is formed and submitted to the oracle. As a result, an event is published consisting of block  $X_j^P(t)$ , which is validated by nodes in  $C_j(t)$ . After reaching consensus for a valid event  $X_j^1(t)$ , it is added to pool  $\mathcal{T}_j$ . As a result, the smart contract in encoding processing block  $C_j^1$  is activated, assigning computation tasks to their corresponding worker nodes.

*Step 2: Worker nodes perform the computations and send the results to the oracle:* Each worker node  $k$  performs the computations and submits the results to the oracle.

*Step 3: Decoding the computation results and obtaining the proposed block:* Upon receiving the computation results from the worker nodes, the oracle publishes an event that includes these results. Subsequently, the nodes within  $C_j(t)$  utilize committee-committee consensus to agree on a valid event, denoted as  $X_j^2(t)$ . This event triggers the smart contract in the decoding processing block  $C_j^2$ . The event  $X_j^2(t)$  is then added to the pool  $\mathcal{T}_j$ . The addition of  $X_j^2(t)$  to  $\mathcal{T}_j$  activates the smart contract in  $C_j^2$ , leading to the generation of the proposed block  $Z_j^P(t)$ , which is subsequently added to the oracle.

### **Stage 2: Validation of Tip Blocks on DAG:**

*Step 1: Computation tasks regarding validating tip blocks on the DAG are assigned to the workers:* Nodes in  $C_j(t)$  first create a super-block  $Z_j^T(t)$  from the tip blocks on the DAG and submit this to the oracle. Subsequently, an event containing the super-block  $Z_j^T(t)$  is published. The nodes then reach a consensus on a valid event, labeled as  $X_j^3(t)$ , which triggers the smart contract in the encoding validation block  $C_j^3$  upon its addition to the pool  $\mathcal{T}_j$ . Following this, the computation tasks assigned are dispatched to their respective worker nodes.

*Step 2: Worker nodes perform the computations and send the results to the oracle:* Each worker node  $k$  performs the computations and submits the results to the oracle.

*Step 3: Decoding the computation results and obtaining the valid super-block:* The oracle collects the computation results from the worker nodes and publishes an event encapsulating these results. Utilizing the committee-committee consensus, nodes in  $C_j(t)$  reach consensus on an event,

denoted as  $\mathcal{X}_j^4(t)$ . This event activates the smart contract in the decoding validation block  $C_j^4$  when is added to pool  $\mathcal{T}_j$ . As a result, a validated super-block  $\mathbf{Z}_j^V(t)$ , comprising the set of valid blocks within  $\mathbf{Z}_j^T(t)$ , is obtained and subsequently added to the oracle.

**Stage 3: Proposed Block Submission and AW Updating:**

*Step 1: Submission of the proposed block:* With  $Z_j^P(t)$  as the proposed block from Stage 1 and  $\mathbf{Z}_j^V(t)$  as the valid super-block from Stage 2, the oracle publishes an event. This event is validated by nodes in  $C_j(t)$  following the committee-committee consensus. The consensus-validated event, denoted as  $\mathcal{X}_j^5(t)$ , is added to the pool  $\mathcal{T}_j$ , triggering the smart contract in the submission block  $C_j^5$ . Subsequently, the proposed block  $Z_j^P(t)$  is submitted to the DAG, creating a new vertex. This vertex establishes directed edges to vertices representing parent blocks in  $\mathbf{Z}_j^V(t)$ , encompassing the valid blocks.

*Step 2: Adding the received proposed blocks from other blockchains to the DAG:* The oracle monitors and publishes an event about the latest blocks added to the DAG by other blockchains. The committee nodes validate this event. Upon validation, triggering the smart contract it adds  $Z_j^P(t)$ , the proposed blocks from other blockchains  $j'$ , to the DAG copy for  $D_j$ .

*Step 3: Updating the AW of blocks on DAG:* Following the committee-committee consensus, nodes in  $C_j(t)$  agree on a valid event,  $\mathcal{X}_j^6(t)$ , published by the oracle. This event concerns the update of the AW of blocks on the DAG. The event  $\mathcal{X}_j^6(t)$  is added to  $\mathcal{T}_j$ . It activates the smart contract  $C_j^5$ , facilitating the update of the AW of the blocks on the DAG and their corresponding status.

**Stage 4: Updating the ledger:**

*Step 1: Updating chain  $\mathcal{Y}_j(t)$ :* Nodes in  $C_j(t)$  create the super-block  $\mathbf{Y}_j^C(t+1)$  from the confirmed blocks on the DAG and then submitted it to the oracle. Following this submission, an event is published. The nodes then reach a consensus on a valid event, denoted as  $\mathcal{X}_j^7(t)$ , which triggers the smart contract in the updating block  $C_j^6$  upon its addition to the pool  $\mathcal{T}_j$ . Then confirmed super-block  $\mathbf{Y}_j^C(t+1)$  is added to chain  $\mathcal{Y}_j$ , resulting in the updated chain  $\mathcal{Y}_j(t+1)$  which is stored by the committee nodes.

*Step 2: Updating event chain  $\mathcal{D}_j$ :* Committee nodes store the ledger  $\mathbf{Y}_j^C(t+1)$  and remove the

pool of temporary events in  $\mathcal{T}_j$ . The set of these removed events is then added to the side event ledger  $\mathcal{D}_j$  as the block  $\mathcal{D}_j(t)$ .

## 4.5 Simulations

This section provides a thorough simulation study of our proposed SPID-Chain system to showcase its excellent performance in terms of throughput and scalability, decentralization, latency and DAG expansion rate, and resistance to double-spending attack.

### 4.5.1 Simulation Setup

To simulate our SPID-Chain system, we employed Substrate [163], a modular framework for building blockchains, to define individual blockchains with distinct consensus mechanisms. These blockchains operate independently while being capable of intercommunication through custom bridging solutions which in our scheme is the DAG ledger. For the DAG structure, we utilized Docker [112] to create a controlled virtual environment, deploying multiple GoShimmer [113] node instances as our blockchains using Docker Compose. This setup allowed us to simulate a network with behavior closely resembling real-world conditions, with the flexibility to adjust parameters as needed. The integration between the Substrate-defined blockchains and the DAG was achieved through a custom-built bridge module, implemented in Rust [164], the primary programming language used for Substrate development. This module utilized Substrate's extensible architecture to facilitate communication between the blockchains and the DAG. Additionally, the Go programming language was employed to interface with the GoShimmer nodes of the DAG, leveraging Go's robust networking capabilities. The bridge module acted as a translator, converting the data formats and consensus signals between the Substrate blockchains and the IOTA-based DAG. This allowed for seamless interoperability, ensuring that transactions could be validated and processed across the different systems without compatibility issues.

In each blockchain we set the number of the committee nodes as  $\frac{n}{10}$ , where  $n$  is the number of worker nodes in each blockchain. Specifically, we employed the libsodium [165] cryptographic

library to implement the verifiable random function (VRF) within our Substrate-based blockchains. Each committee node generates a VRF output using its private key and a shared random seed, synchronized across the network using the network time protocol (NTP) to ensure consistent epoch times. The seed is distributed securely to all nodes at the start of each epoch through a decentralized broadcast protocol.

In the simulations, honest blockchains adhere to both intra- and inter-consensus protocols. In particular, during Stage 3, they follow established guidelines for selecting parent blocks and issuing valid blocks. We denote the per-minute block issuance rate by honest blockchains during this stage as  $\gamma_h$ . On the other hand, adversarial blockchains follow the intra-consensus protocol but deviate during Stage 3 by consistently issuing invalid blocks. Moreover, in selecting parent blocks, adversarial blockchains adopt the *orphanage attack* strategy that involves choosing a parent with the oldest possible generation age and, when feasible, prioritizing their own blocks at the DAG’s tip. We denote the per-minute block issuance rate by adversarial blockchains during Stage 3 as  $\gamma_a$ .

Consequently, the total block issuance rate per minute of the DAG is given by  $\gamma = \gamma_a + \gamma_h$ , and the spamming rate of the adversarial blockchains is defined as  $\mu = \frac{\gamma_a}{\gamma}$ . The critical spamming rate is defined as  $\mu_{\text{crit}} = \frac{K-1}{K}$ , where  $K$  is the number of blocks selected for processing in Stage 3 of the protocol. If  $\mu > \mu_{\text{crit}}$  then the tip pool size grows rapidly with  $\mu$ , as demonstrated in [109] for the IOTA-type DAG ledger.

Each simulation runs for a duration of 5 minutes, during which 500 blocks are generated at a rate of  $\gamma = 100$  blocks per minute. These blocks are subsequently attached to the DAG.

Additionally, for a given value  $\lambda$ , we randomly select  $\lambda n$  worker nodes to form the set of stragglers  $\mathcal{S}$ . A straggler never returns any results while If a node is not a straggler, it always returns the computation results correctly and on time.

We utilized the BigQuery [166] public Ethereum dataset to obtain samples of real-world blockchain transactions. To generate invalid blocks, we modified the account balances in a subset of transactions so that they exceed the maximum allowable balance. These altered transactions were then grouped with unmodified, valid transactions to form invalid blocks. In our simulation, while

a valid block contains just valid transactions, each invalid block contains a mixture of valid and invalid transactions. We set the number of accounts in each blockchain to  $M = 1000$ . Moreover, for each simulation, 20% of the blockchains are designated as adversarial with varying spamming rate.

Finally, each result is the average of 10 runs. We set both the task completion timeout for workers and voting timeout for committee nodes to 500 milliseconds, with a round stage occurring when all tasks are completed. The epoch transition happens when nodes in a blockchain have processed the entire task list. We also set the bandwidth to 20 Mbps and imposed a latency of 100 ms on all communication links in all simulations.

## 4.5.2 Results

### Throughput and Scalability

In this section, we will investigate the throughput and scalability of our proposed scheme, focusing on two distinct aspects: intra-consensus and inter-consensus. Throughput is a critical metric that assesses the speed and accuracy with which blocks are processed within a system. Scalability, on the other hand, is an essential factor in determining a system's capacity to manage growing workloads while maintaining efficient performance.

**Intra-consensus throughput:** Intra-consensus throughput is defined as the average number of blocks processed by a scheme per minute. The intra-consensus throughput focuses on the processing speed and accuracy of blocks within each individual blockchain. To measure intra-consensus throughput, we calculate the *average* number of blocks correctly generated by honest blockchains within the network per minute. Notably, the intra-consensus throughput remains unaffected by the spamming rate of adversarial blockchains, as blocks from these sources are deemed invalid and thus excluded from the calculations. Furthermore, since blocks generated by honest blockchains are always considered valid, there is no need to assess the accuracy of these blocks.

In our throughput simulations, we define a block as processed once its AW exceeds 67% and it's confirmed. We set up  $N = 10$  blockchains, each with  $n = 100$  worker nodes, varying the percentage of stragglers  $\lambda \in \{10\%, 30\%\}$  and the rate of incoming blocks. Our findings, illustrated in Fig.

4.2, show that blockchains using the SPID-Chain scheme process transactions up to 11 blocks per minute before the slope of the throughput begins slightly to decline, particularly at a higher straggler rate of 30%. In contrast, networks without encoding experience a steady decline in throughput.

Moreover, intra-consensus throughput is generally lower in networks without encoding. The use of Polar encoding in SPID-Chain helps distribute computations across nodes, mitigating the negative impact of stragglers by offloading tasks to faster workers. This efficiency, however, comes at the cost of increased data transmission bandwidth between committee and worker nodes.

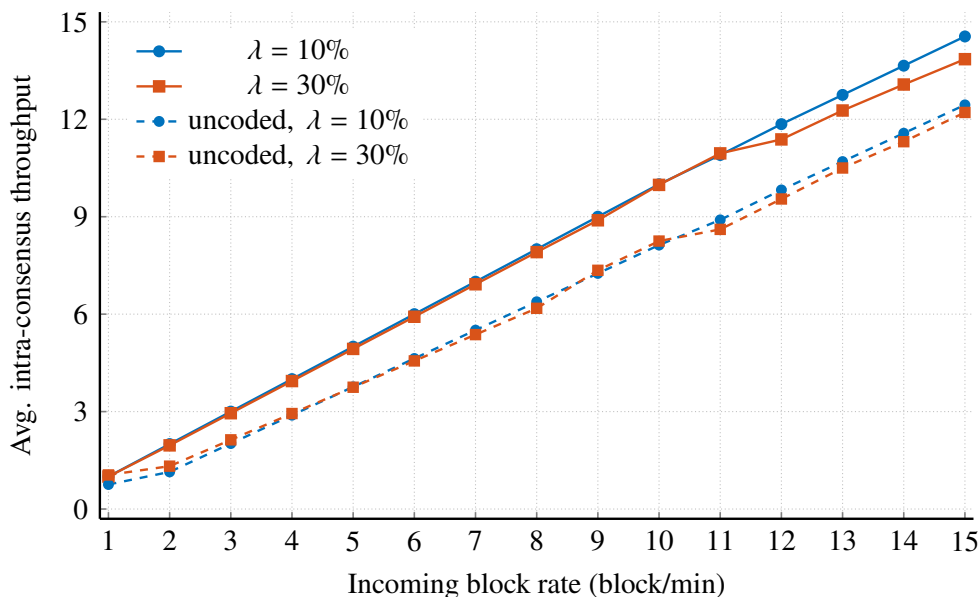


Figure 4.2: Intra-consensus throughput with  $N = 10$ ,  $n = 100$ , and for varying straggler worker percentages  $\lambda$  and incoming block rates.

**Intra-consensus scalability:** Intra-consensus scalability refers to a blockchain system’s ability to increase the number of nodes within each blockchain and it is illustrated in terms of the intra-consensus versus the number of nodes in each blockchain. By setting incoming block rates at 12 blocks per minute with  $\lambda = 10\%$ , and varying the number of blockchains  $N \in \{5, 15\}$ , the system demonstrates how scalability is affected by node count. Fig. 4.3 shows the intra-consensus scalability. Systems using the Polar encoding scheme successfully manage the addition of the worker nodes, by improving work distribution among the worker nodes. In contrast, systems lacking encoding experience a decrease in throughput as more nodes lead to slower computations and

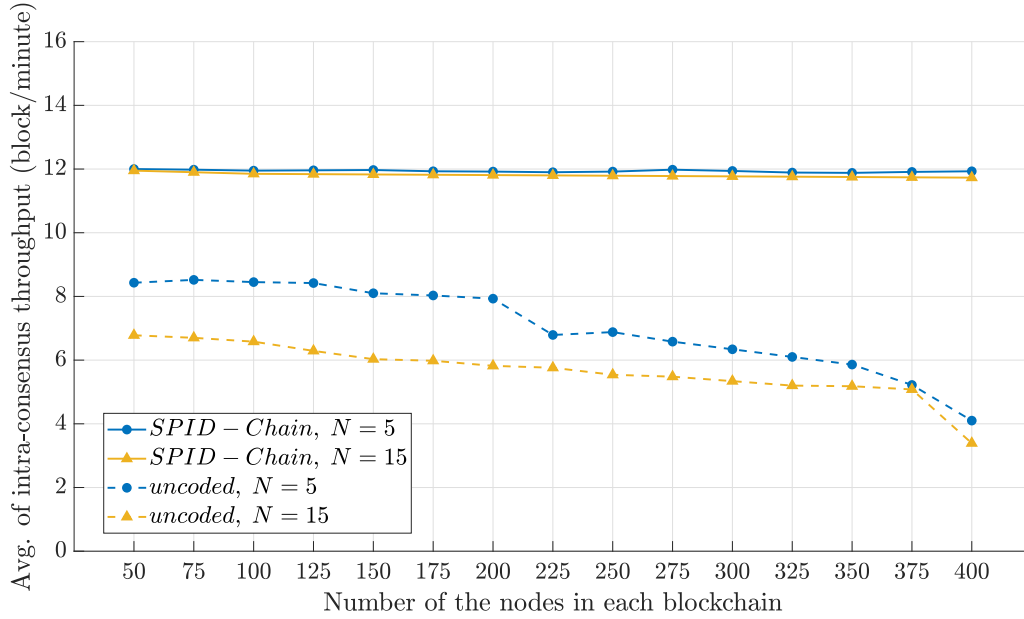


Figure 4.3: Intra-consensus scalability with  $\lambda = 10\%$ , 12 blocks per minute for rate of incoming blocks, and for varying numbers of nodes within each blockchain and different numbers of blockchains  $N$ .

extended protocol stages.

**Inter-consensus throughput:** The inter-consensus throughput measures the rate at which blocks are processed on the DAG ledger. For evaluating the inter-consensus throughput, it is crucial to consider both the accuracy, which indicates the correctness of decisions made during block processing on the DAG, and the time required for processing blocks. Therefore, we count the number of the correctly confirmed blocks on the DAG per minute.

Figure 4.4 illustrates the inter-consensus throughput with settings of  $\lambda = 10\%$ ,  $K = 2$ , resulting in a critical spamming rate  $\mu_{\text{crit}} = 50\%$ , and varying spamming rates  $\mu \in \{10\%, 55\%\}$ . Throughput improves as the rate of incoming blocks increases, leveraging the DAG structure where new blocks increase the AW of existing ones, thereby facilitating faster confirmation. This mechanism underscores the strength of DAG-based ledgers in scaling confirmation and processing speeds. Notably, at a low spamming rate of  $\mu = 10\%$ , throughput significantly rises as the block rate exceeds 8 per minute. On the other hand, at  $\mu = 55\%$ , despite still experiencing an increase, throughput growth is less pronounced due to adversarial actions that modify the DAG’s structure, leading

to fewer correct block confirmations. Nonetheless, even under higher spamming conditions of  $55\% = \mu > \mu_{\text{crit}}$ , SPID-Chain shows an overall throughput increase.

Similarly, for the systems using dashed lines where encoding is absent, we observe the same trend of increased throughput with a rising rate of incoming blocks. However, this effect is much less pronounced compared to systems with Polar encoding, to the extent that it diminishes the advantages of using a DAG ledger. Specifically, the use of Polar encoding enables honest blockchains to process blocks more efficiently by distributing the workload, which helps them counteract the spamming tactics of adversarial nodes. Specifically, in Stages 1 and 2 of the SPID-Chain, the implementation of distributed coding accelerates the processing of each honest blockchain’s proposed block and the tip blocks on the DAG. This acceleration is evident in Stage 3, where it significantly enhances the inter-consensus throughput. The faster selection and processing of the proposed block and its parents from Stages 1 and 2 allow for quicker submissions to the DAG in Stage 3, thereby increasing the inter-consensus throughput.

In contrast, without encoding, it takes longer to process blocks and attach them to the DAG, allowing adversarial blockchains to attach their blocks more strategically, leading to the confirmation of more invalid blocks and, therefore, lowering the accuracy.

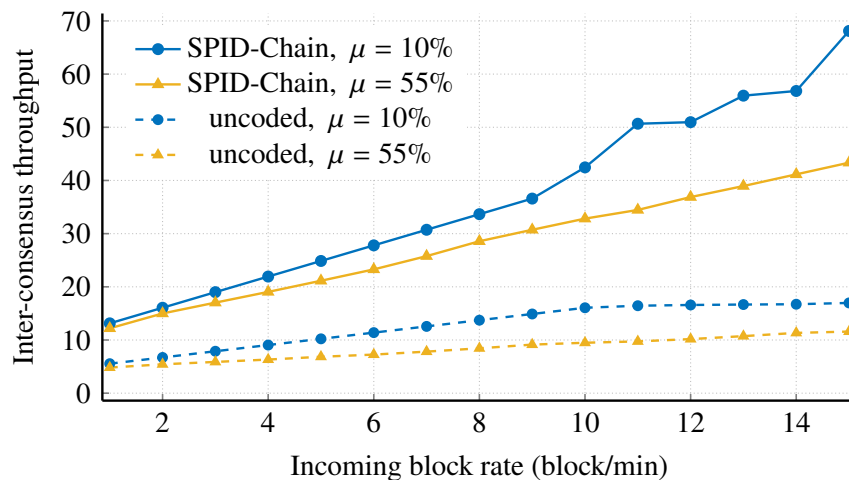


Figure 4.4: Inter-consensus throughput with  $\lambda = 10\%$ ,  $N = 10$ ,  $K = 2$ ,  $\mu_{\text{crit}} = 50\%$ , and for varying rates of incoming blocks and spamming rates.

**Inter-consensus scalability:** Inter-consensus scalability pertains the ability of a system to

handle an increased number of interconnected blockchains within a DAG structure, crucial for managing more blocks across multiple blockchains. We maintained a constant block arrival rate of 10 blocks per minute with specific parameters  $\mu = 20\%$ ,  $\lambda = 10\%$ ,  $K = 2$ ,  $\mu_{\text{crit}} = 50\%$ ,  $\eta = 67\%$  and varied both the number of blockchains  $N$  and the worker nodes per blockchain  $n \in \{100, 200\}$ .

Figure 4.5 illustrates the inter-consensus scalability of the SPID-Chain. As depicted, the system's performance improves with the addition of more blockchains, marked by increased inter-consensus throughput. This enhancement is due to the faster accumulation of blocks within the DAG and the accelerated AW of these blocks, which expedites the confirmation process a key feature of DAG-based ledgers. However, for the solid lines represent the SPID-Chain model utilizing Polar encoding it is observed that as the number of nodes within each blockchain rises, so does the system's throughput. This improvement is largely because Polar encoding streamlines the block processing and distribution of computational tasks among nodes, making the system more efficient. On the other hand, the dashed lines represent systems without encoding. Here, although throughput increases with more blockchains, the rise is less marked. This suggests that networks lacking encoding fail to fully leverage the inherent scalability benefits of DAG ledgers. The longer processing times result in fewer blocks being attached to the DAG, compromising the ledger's scalability advantage. Moreover, in systems without encoding, increasing the number of worker nodes does not enhance throughput but instead decreases it, due to longer times required to compile results, complete computations, achieve consensus, and process blocks.

## **Decentralization**

In the context of blockchain interoperability and DAG-based ledger systems, decentralization is a crucial aspect that extends beyond mere block throughput. It is essential to consider the distribution of confirmed blocks across the DAG, as a system with high throughput but concentrated block confirmations among a subset of blockchains may not truly embody the decentralized ethos of blockchain technology. A scheme that ensures a more uniform distribution of confirmed blocks across participating blockchains fosters a more resilient and decentralized network, reducing the

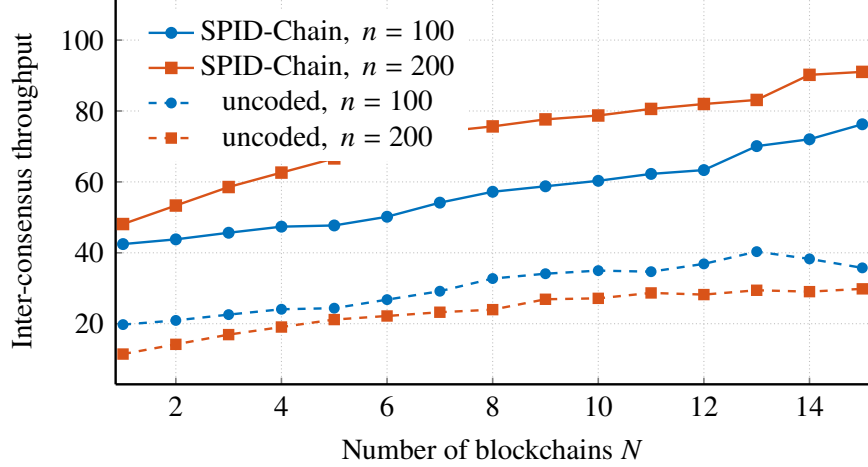


Figure 4.5: Inter-consensus scalability for varying numbers of blockchains  $N$  and worker nodes within each blockchain  $n \in \{100, 200\}$  with fixed incoming block rate 10 blocks per min,  $\mu = 20\%$ ,  $\lambda = 10\%$ ,  $K = 2$ , and  $\mu_{\text{crit}} = 50\%$ .

risk of centralization and the potential for collusion among a few dominant blockchains.

We employ the Gini coefficient [55] as a measure of the decentralization of our network. The Gini coefficient is defined as:

$$I(t) = \frac{\sum_{i=1}^N \sum_{j=1}^N |\phi_i(t) - \phi_j(t)|}{2N \sum_{j=1}^N \phi_j(t)}. \quad (4.11)$$

Here,  $\phi_j(t)$  represents the number of blocks confirmed by chain  $j$  at epoch  $t$ . Smaller variations among  $\phi_j(t)$  result in a lower Gini coefficient, indicating better decentralization, as it implies a more even distribution of block confirmations across the network. As measured in [55], the Gini coefficients for the Bitcoin and Ethereum networks average 0.8 and 0.5, respectively. These two blockchains, known as dominant and widely recognized benchmarks, can serve as standards for evaluating the decentralization performance of various schemes.

We simulate our scheme for different values of  $K$  and varying the spamming rate  $\mu$  of the adversarial. For all cases, we set  $N = 10$ ,  $n = 100$ , and  $\lambda = 10\%$ . Fig. 4.6 shows the decentralization of SPID-Chain. As we see for all the different cases and scenarios the proposed scheme shows good decentralization comparing to the Gini coefficients of Bitcoin and Ethereum. Moreover, as the parameter  $K$  increases, there is a corresponding decrease in the maximum Gini coefficient

across the bins in each subplot, suggesting an enhancement in the decentralization of the network. Specifically, for each value of  $K$ , when the spamming rate  $\mu$  surpasses the critical threshold ( $\mu_{\text{crit}}$ ), decentralization begins to deteriorate as evidenced by an increasing Gini coefficient. However, as  $K$  increases, this degradation becomes progressively less pronounced.

The reason is the system's ability to validate more blocks through honest blockchains, which in turn select a greater number of parents for the blocks they add to the DAG  $K$ , effectively counters the orphanage strategy employed by adversarial blockchains. This strategy typically involves adversarial blockchains preferentially attaching their blocks to previous blocks of their own making. Our simulations demonstrate that increased  $K$  values mitigate this behavior, leading to a more decentralized and secure network that upholds the integrity of the ledger against such adversarial tactics.

## Security

**Latency and DAG expansion rate:** In the domain of blockchain interoperability, the two primary metrics of importance are the inter-chain block finality time and the DAG expansion rate, which basically is the evolution of the tip pool size over the time.

Inter-chain block finality time is critical for assessing the latency in cross-chain transactions, indicating the time it takes for a block to be irreversibly recorded on the DAG ledger. A shorter finality time suggests a faster system, which is crucial for user experience and the swift settlement of transactions.

The DAG expansion rate provides insights into the network's structural growth by measuring how quickly the DAG's tip pool size increases with contributions from various blockchains. This rate is key to understanding the network's activity level and its ability to scale efficiently. Proper management of the tip pool size is vital to avoid congestion and maintain transaction processing efficiency, which in turn ensures the network's security.

For an effective DAG ledger, the size of the tip pool should remain relatively stable and small over

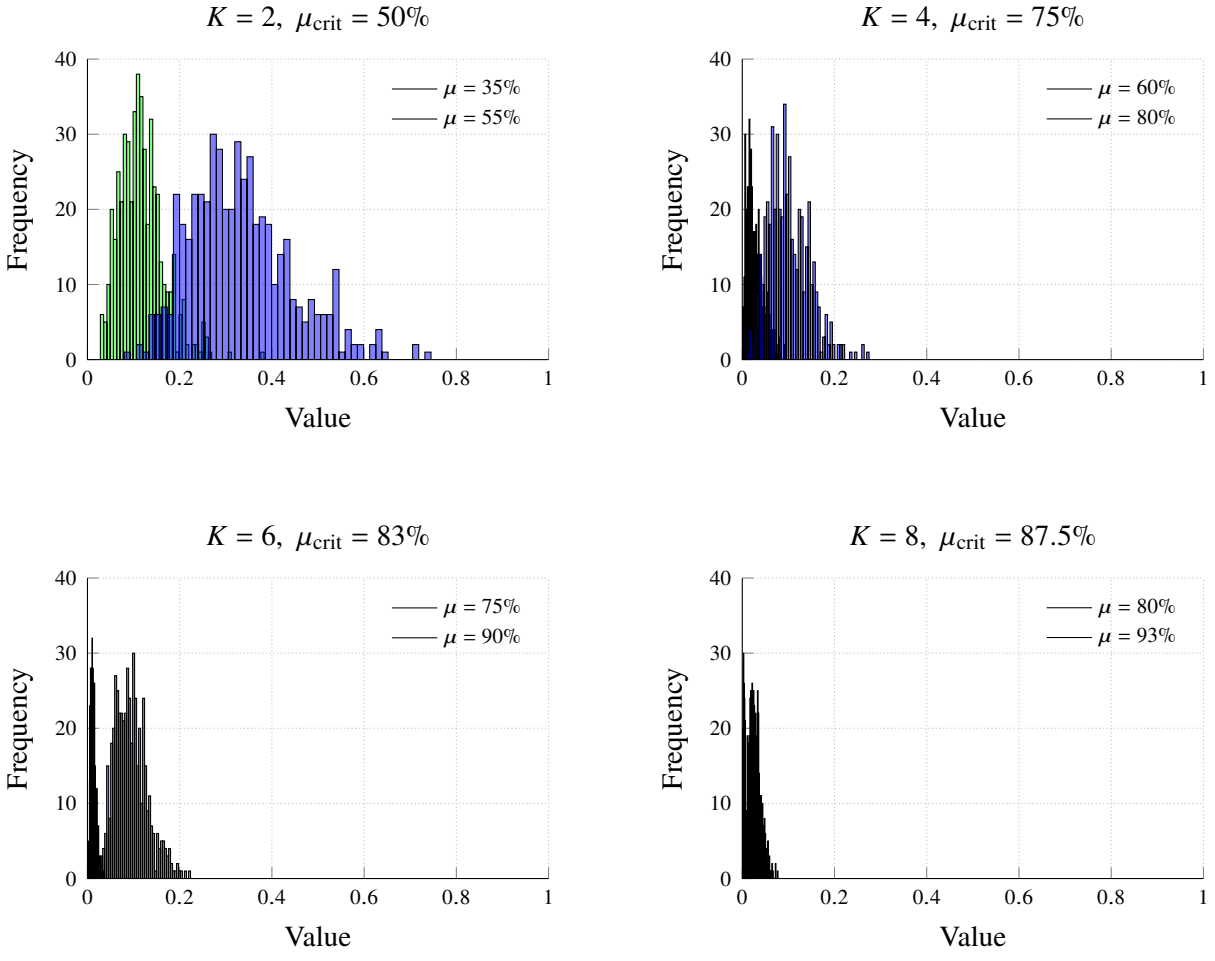


Figure 4.6: Network decentralization measured by Gini coefficient for different  $K$  and  $\mu_{\text{crit}}$  values and spamming rates ( $\mu$ ), with  $N = 10$ ,  $n = 100$ , and  $\lambda = 10\%$ .

time, showing less sensitivity to the spamming rates imposed by adversarial blockchains. An increase in the tip pool size reduces the likelihood of blocks at the tip being selected for validation, thereby extending the time to finality. In scenarios where the tip pool size grows exponentially—referred to as over-inflation—it leads to a higher probability of blocks becoming orphaned, as discussed in [167].

To assess the latency and DAG expansion rate, i.e., tip pool size, in our system, as well as to explore the impact of the critical point, we simulated a network consisting of  $N = 10$  blockchains, each with  $n = 100$  worker nodes. Among these nodes,  $\lambda = 10\%$  were identified as stragglers. We set  $\eta = 67\%$  and tested two settings:  $K = 2$  with  $\mu \in \{35\%, 55\%\}$  and  $K = 4$  with  $\mu \in \{60\%, 80\%\}$ . This span covers both below and above the critical spamming rates of  $\mu_{\text{crit}} = 0.5$  for  $K = 2$  and  $\mu_{\text{crit}} = 0.75$  for  $K = 4$ .

For each  $\mu$  value, the network operated for 12 minutes, during which we monitored the confirmation status of blocks on the DAG. Fig. 4.7 illustrates the tip pool size and provides a scatter plot of the inter-chain block finality times. Results show that higher  $\mu$  values lead to an increased tip pool size and longer confirmation times for blocks. Notably, with both  $K = 2$  and  $K = 4$ , the tip pool size and inter-chain block finality times exhibit increased vulnerability when  $\mu > \mu_{\text{crit}}$ , as evidenced by an expansion in the tip pool size and extended finality times for the blocks. However, as shown in Fig. 4.7 for  $K = 4$ , the adverse effects on the tip pool size and finality times are less severe when  $\mu$  exceeds  $\mu_{\text{crit}}$ . Therefore, it is recommended that for enhanced DAG ledger health and reduced finality times, larger values of  $K$  should be considered.

**Double Spend Resistant Analysis:** In blockchain systems, a double spend attack refers to a scenario where an attacker successfully spends the same digital currency or asset twice. This type of attack exploits vulnerabilities in the network’s consensus mechanism or propagation delays to manipulate transaction records.

To simulate double-spend scenarios within our SPID-Chain framework, we devised a specific method for creating double-spend blocks. Each double-spend block was constructed to contain transactions that are valid in isolation—meaning the sender possesses adequate tokens for the

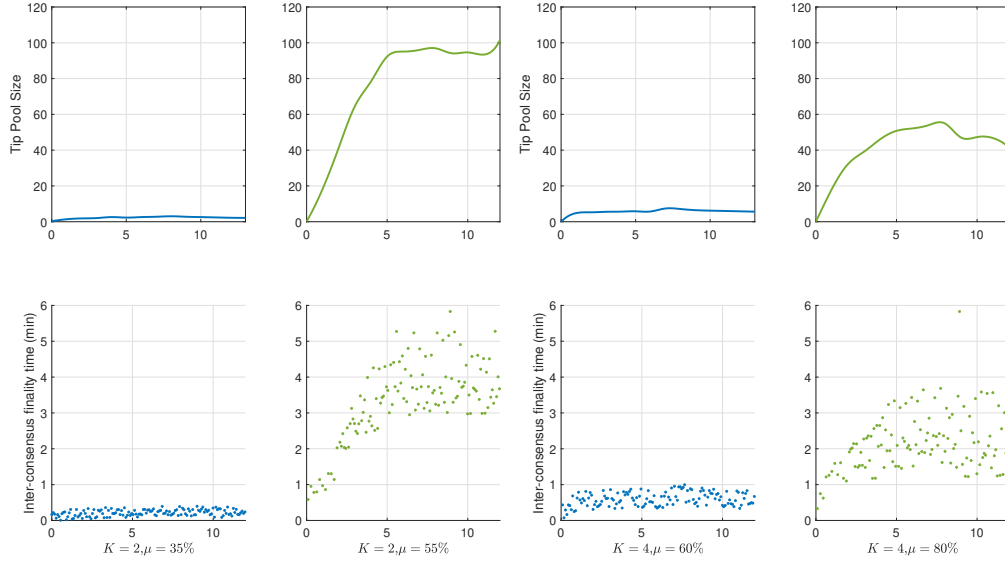


Figure 4.7: Impact of varying spamming rate ( $\mu$ ) on DAG ledger health, demonstrated by tip pool size and inter-chain block finality time. Simulation parameters:  $N = 10$  blockchains,  $n = 100$  worker nodes with  $\lambda = 10\%$ ,  $\eta = 67\%$ ,  $K = 2$ ,  $\mu_{\text{crit}} = 50\%$  and  $\mu \in \{35\%, 55\%\}$ ,  $K = 4$ ,  $\mu_{\text{crit}} = 75\%$  and  $\mu \in \{60\%, 80\%\}$ .

transaction, thus each transaction adheres to standard validation protocols. However, the double-spend characteristic was introduced by replicating the same transaction, with identical signatures and identifiers, across two different blocks. Specifically, among the 100 double-spend blocks prepared for the simulation, we organized these into 50 pairs, with each pair containing one replicated transaction while maintaining unique non-duplicate transactions in the remainder of the blocks.

To integrate these double-spend blocks into our network simulation, we attached 100 double-spend blocks at random points within the DAG, in addition to the 300 regular blocks. This approach created a diverse tip pool that included both double-spend and regular transaction blocks. The detection mechanism within the SPID-Chain was tasked with identifying and labeling the double-spend transactions based on discrepancies observed in the transaction records across the DAG by the committee nodes. Specifically, In Stage 2, committee nodes while assembling the super-block  $\mathbf{Z}_j^T(t)$  during an epoch  $t$ , exclude any double-spend blocks detected. Upon submitting their proposed block  $Z_j^P(t)$  to the DAG in Stage 3, they label any double-spend blocks within  $\mathbf{Z}_j^T(t)$ . A double-spend block is considered detected once the block  $Z_j^P(t)$  is confirmed as  $Z_j^C(t)$ .

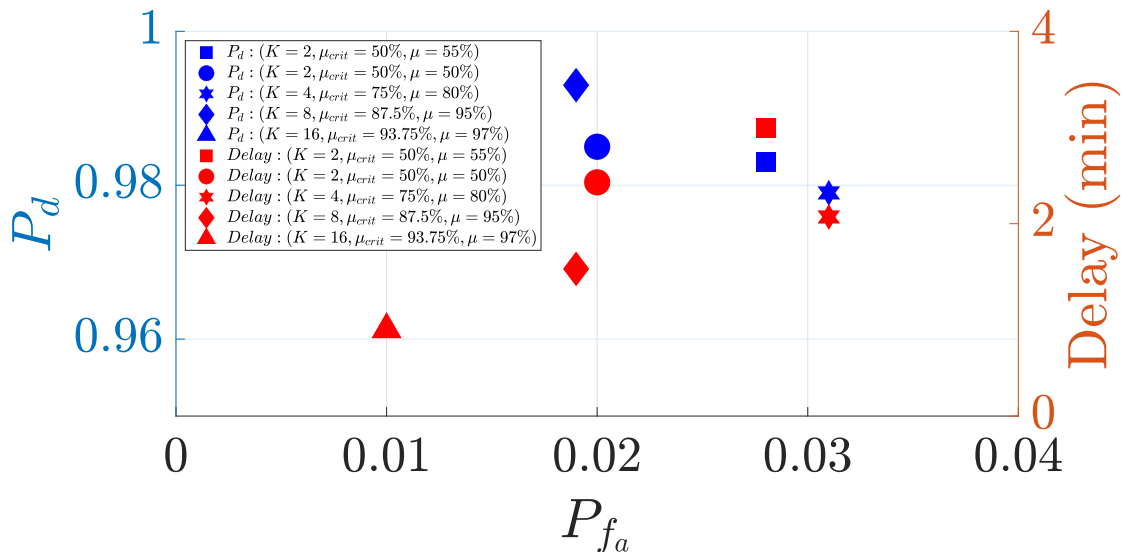


Figure 4.8: Performance of SPID-Chain in detecting double-spend blocks with  $N = 10$ ,  $\lambda = 10\%$ , and  $\eta = 67\%$ .

We set  $N = 10$ ,  $\lambda = 10\%$ ,  $\eta = 67\%$ , and varied the spamming rate of adversaries,  $\mu$ , and the number of blocks selected for processing from the tip pool,  $K$ . The probability of false alarm,  $P_{fa}$ , is defined as the percentage of regular blocks mistakenly identified as double-spend blocks. Conversely, the probability of detection,  $P_d$ , refers to the percentage of correctly identifying actual double-spend blocks. We also measured the speed of identification as the delay where we measure after all the 400 blocks are processed.

The results, displayed in Fig. 4.8, demonstrate that the SPID-Chain effectively identifies double-spend blocks across all  $K$  values with a low  $P_{fa}$ . Larger  $K$  values slightly increase  $P_d$  and reduce identification delays, which aligns with expectations as processing more blocks per chain increases detection chances and speed.

It is crucial to note the simulation also accounted for adversaries operating above the critical spamming rate, significantly challenging the DAG's topology and the process of invalidating double-spend attacks. Despite this, SPID-Chain effectively resisted the attacks. For instance, at  $K = 2$  and  $\mu = 55\%$ , despite the longer detection time compared to the critical spamming rate of  $\mu = 50\%$ , the system still managed to detect double-spend blocks efficiently with acceptable delays.

## 4.6 Conclusion

The proposed SPID-Chain framework offers a new approach to achieving interoperability within Web3 ecosystems. By structuring a DAG of blockchains, SPID-Chain uses this DAG for inter-consensus processes while incorporating EDSC and Polar codes for intra-consensus operations. This design effectively tackles the scalability, security, and decentralization issues that are common in existing blockchain interoperability solutions. Our extensive simulations confirm that SPID-Chain performs robustly, demonstrating its potential for enabling secure and seamless interactions across multiple blockchains.

## Chapter 5: ZK-HybridFL: Zero-Knowledge Proof-Enhanced Hybrid Ledger for Federated Learning

### 5.1 Introduction

Federated Learning (FL) has emerged as a promising paradigm for collaboratively training models across distributed data silos while preserving privacy [168]. However, traditional FL architectures typically rely on centralized coordinators, potentially creating single points of failure and trust bottlenecks [169]. To overcome these limitations, decentralized FL frameworks have been proposed that leverage distributed ledger technologies (DLTs) for trustless coordination [170].

A variety of blockchain-based approaches illustrate the potential of decentralizing FL. For instance, BlockFL [171] optimizes secure model exchanges and rewards by adjusting the block generation rate, while PIRATE [172] implements a sharding-based design to achieve Byzantine resilience through secure gradient aggregation. Other notable efforts include adaptive FL methods that reduce communication overhead by up to 50% [173], incentive mechanisms grounded in reputation systems [174], PoS-based resource-saving algorithms [175], Top<sub>k</sub> compression for limiting communication costs [176], and blockchain-empowered frameworks for 5G-enabled UAVs [177]. Further studies have analyzed energy consumption in blockchain networks [178], latency optimization [171], and integration with mobile edge computing [179], while foundational research has established theoretical bounds for resource-constrained FL settings [180] and proposed reputation-oriented incentive schemes for preserving IoT data privacy [181].

Despite these advances, significant obstacles remain. Current frameworks often struggle to balance scalability, security, and privacy when operating at large scale or in diverse, adversarial environments [182]. Synchronous systems (e.g., those relying on Proof of Work or Practical Byzantine Fault Tolerance) can become bottlenecks under straggler nodes, while solutions that rely

on public datasets for validation risk data exposure and fail to accommodate the heterogeneity of real-world data [183]. Moreover, advanced gradient inversion techniques [184], label inference attacks [185], and dis-aggregation methods [186] highlight the difficulty of protecting participant information purely through naive data-sharing defenses. Although differential privacy [187] can mitigate certain risks, it does not resolve the fundamental challenge of publicly verifying the integrity of local model updates at scale [188].

### 5.1.1 State-of-the-art and Challenges

Blade-FL [15] removes the need for a centralized server by integrating blockchain directly into the FL workflow. In each epoch, nodes train their local models on mini-batches, then sign and broadcast the updates as blockchain transactions. These updates are verified against a public dataset based on a loss threshold and are aggregated—by averaging the validated updates—into a global model. Subsequently, nodes compete to mine a new block via Proof of Work (PoW), which records both the verified transactions and the aggregated model. Once the block is validated network-wide, the updated global model is adopted for the next training round. This cycle repeats until convergence.

ChainFL [16] adopts a hierarchical blockchain structure of node, subchain, and mainchain layers to facilitate FL without a central server. A task is initiated via a smart contract on a Directed Acyclic Graph (DAG) ledger, broadcasting the request across shards. Within each shard, the Subchain Leader Node (SLN) distributes the current global model to participating nodes, collects their locally trained updates, and aggregates them—weighted by dataset sizes—using the Raft consensus mechanism before committing the aggregated result to that shard’s blockchain. The SLN then submits this shard-level model to the DAG, where top models from different shards are evaluated on a public dataset to form the next global model. This iterative process of local training, aggregation, and cross-shard validation continues until the smart contract conditions signal completion of the FL task.

Both Blade-FL and ChainFL introduce valuable decentralization strategies but face critical limitations in practice. Blade-FL’s reliance on PoW leads to heavy computational overhead and high energy consumption, while the use of a public validation dataset raises privacy concerns and risks

adversarial manipulation. ChainFL’s hierarchical, sharded structure, though beneficial for scaling, still depends on centralized public dataset evaluations and suffers from cross-shard synchronization overhead that can allow stale or malicious updates to proliferate. Motivated by these shortcomings, our work proposes a novel framework that integrates a DAG-based ledger with sidechains and employs Zero-Knowledge Proofs (ZKPs) for privacy-preserving, on-device model validation. This approach not only eliminates the need for public datasets but also boosts efficiency and scalability by ensuring only genuine, high-quality updates are accepted into the global model.

At a broader level, existing solutions also highlight the unmet need for robust and efficient methods of verifying local model updates without compromising user data. Conventional methods often resort to public datasets for legitimacy checks [183], [189], exposing privacy vulnerabilities and undermining real-world representativeness. Advanced cryptographic protocols, such as ZKPs [190], can address privacy and correctness simultaneously, but current implementations often support only specific model types (e.g., linear regression [191]) or suffer from high computational overheads [192]. These drawbacks underscore the need for a more scalable, generalizable solution that keeps participant data private while enabling secure, public verification of model contributions.

In this chapter, we introduce ZK-HybridFL a novel decentralized FL architecture that addresses these gaps through an innovative ledger design and cryptographic validation pipeline. Our framework combines a DAG for high-throughput storage of model updates with sidechains dedicated to event-driven smart contracts (EDSCs). By embedding ZKPs in the sidechain-based contracts, we create a method for verifying training correctness without disclosing sensitive information, thereby eliminating the overhead and security pitfalls of traditional solutions.

### 5.1.2 Contributions

Our primary contributions are as follows:

1. **Hybrid Ledger System:** We propose a DAG-based ledger for scalable storage of model updates, augmented by sidechains running EDSCs to manage consensus and validation tasks. This structure alleviates bottlenecks inherent in purely PoW- or DAG-based systems.

2. **ZKP-Driven Secure Validation:** By integrating ZKPs into EDSCs, our framework enables public verification of inference correctness without exposing private test data or imposing prohibitive computational costs.
3. **Experimental Validation:** Through extensive simulations, we demonstrate that our approach outperforms Blade-FL [15] and ChainFL [16] in accuracy, convergence speed, and resilience, even in the presence of adversarial or lazy nodes.

The remainder of the chapter is organized as follows. Sec. 5.2 introduces the ZKP-based consensus mechanism and DAG ledger for decentralized FL. Sec. 5.3 describes the event-driven smart contracts and oracle-assisted sidechain. Sec. 5.4 outlines the ZK-HybridFL workflow and analysis. Sec. 5.5 presents simulation results and comparisons with existing schemes, and Sec. 5.6 concludes the chapter with future research directions.

## 5.2 ZKP-based Consensus Mechanism of ZK-HybridFL

In this section, we first present the use of ZKPs for privacy-preserving model validation. Next, we explain the DAG ledger, which manages scalable and decentralized model updates. Finally, we discuss the challenge mechanism for verifying blocks and resolving conflicts to ensure system integrity.

### 5.2.1 ZKP

#### **Motivation of using private data for model validation**

When selecting between a public test dataset and private test datasets secured via ZKP in decentralized federated learning (FL), opting for private test datasets offers significant advantages in both privacy and model quality. Utilizing a public test dataset simplifies evaluation by providing a uniform benchmark for all nodes, which facilitates straightforward comparisons. However, the public and static nature of such datasets may result in models being validated on a limited variety of unseen data, potentially restricting their robustness and adaptability [187]. Additionally, as nodes

continuously refine their local models to perform well on the public test set, the evolving patterns in their loss or accuracy can inadvertently expose information about their private training data through techniques like membership inference or model inversion [193], [194]. In contrast, employing private test datasets with ZKPs ensures that each local model is assessed using diverse and private test samples. This approach enhances exposure to a broader data distribution, thereby improving the global model’s quality and generalizability while simultaneously protecting data privacy.

### The ZKP process

A ZKP is a cryptographic technique that enables one party (the prover) to convince another party (the verifier) of the validity of a statement without revealing any information beyond the fact that the statement is valid. In computation-related applications, the prover and verifier agree on a specific algorithm or function. The goal is for the prover to demonstrate to the verifier that the output of the algorithm or function is indeed the result of a particular input, without disclosing any additional details about the input or process.

In ZK-HybridFL, ZKPs enable a node  $j$  (prover) to demonstrate to other nodes  $j'$  (verifiers) that its inference output  $\mathcal{Y}_j^t$  at time step  $t$  is correctly derived by applying its publicly available model  $\mathbf{W}_j^t$ , trained during epoch  $t$ , to its private test batch  $\mathcal{D}_j^{t,\text{test}}$  using the inference algorithm  $\mathcal{I}$  without revealing any details about  $\mathcal{D}_j^{t,\text{test}}$  or internal computations. Recall that each node  $j$  maintains a local dataset  $\mathcal{D}_j^t$ , and a training mini-batch  $\mathcal{D}_j^{t,\text{train}}$  is used to obtain the model  $\mathbf{W}_j^t$ . The test mini-batch  $\mathcal{D}_j^{t,\text{test}}$  is drawn from the remaining data, i.e.,  $\mathcal{D}_j^{t,\text{test}} \subset \mathcal{D}_j^t \setminus \mathcal{D}_j^{t,\text{train}}$ .

To achieve ZKP, a predefined, immutable program called the circuit, is created based on the given inference algorithm  $\mathcal{I}$ . Once compiled, this circuit executes a sequence of unforgeable operations that generate both the predicted output  $\mathcal{Y}_j^t$  and its associated loss value  $\mathcal{L}(\mathcal{D}_j^{t,\text{test}}, \mathcal{Y}_j^t)$ , which measures the quality of model  $\mathbf{W}_j^t$ . In the process, it produces explicitly defined, immutable intermediate results, referred to as witnesses and denoted by  $\mathcal{U}_j^t$ . To ensure the integrity of this computation, a verifier must check the consistency of the inputs  $\mathcal{D}_j^{t,\text{test}}$  and  $\mathbf{W}_j^t$ , the intermediate results  $\mathcal{U}_j^t$ , and the outputs  $\mathcal{Y}_j^t$  and  $\mathcal{L}(\mathcal{D}_j^{t,\text{test}}, \mathcal{Y}_j^t)$ . In particular, the prover node  $j$  and the verifier

node  $j'$  follow the sequence of actions outlined below:

1. **Commit.** At the beginning of epoch  $t$  the trainer binds *both* its freshly-trained weight tensor  $\mathbf{W}_j^t$  and its private test batch  $D_j^{t,\text{test}}$  by posting polynomial commitments  $C_j^{t,\text{model}} = \text{Commit}(\mathbf{W}_j^t)$  and  $C_j^{t,\text{test}} = \text{Commit}(D_j^{t,\text{test}})$  to the side-chain.<sup>1</sup>
2. **Key availability.** A one-time Powers-of-Tau ceremony (run off-chain by the oracle committee) produces a *circuit-specific* proving key  $\text{pk}$  and a public verification key  $\text{vk}$ . Because the inference circuit is fixed for the entire task,  $\text{pk}$  and  $\text{vk}$  are reused by *every* node and for *every* epoch; only the private witness  $(\mathbf{W}_j^t, D_j^{t,\text{test}})$  changes.
3. **Proof generation.** Using the proving key  $\text{pk}$  and its private witness  $(\mathbf{W}_j^t, D_j^{t,\text{test}})$ , node  $j$  evaluates the circuit to obtain  $\mathcal{U}_j^t$  (the full witness) and the public outputs  $\mathcal{Y}_j^t, \mathcal{L}_j^t$ . It then computes the non-interactive Groth16 proof  $\Pi_j^t = \text{Prove}(\text{pk}, \mathcal{U}_j^t)$ , where all Fiat–Shamir challenges are derived deterministically from  $\text{H}(\text{blockHash} \parallel C_j^{t,\text{model}} \parallel C_j^{t,\text{test}})$ , so no live randomness exchange is needed. Finally, trainer  $j$  broadcasts  $Z_j^t = (\Pi_j^t, \mathcal{Y}_j^t, \mathcal{L}_j^t, C_j^{t,\text{model}}, C_j^{t,\text{test}})$ .
4. **Verification.** Any verifier  $j'$  fetches the verification key  $\text{vk}$  from the side-chain and checks  $\text{Verify}(\text{vk}, \Pi_j^t, C_j^{t,\text{model}}, C_j^{t,\text{test}}, \mathcal{Y}_j^t, \mathcal{L}_j^t) = 1$ . A result of 1 certifies that the public outputs  $\mathcal{Y}_j^t, \mathcal{L}_j^t$  were produced *exactly* by the committed model and test data; otherwise the update is rejected. Once verification succeeds, a **ProofOK** event is emitted, letting the contribution enter the DAG.

The keys  $\text{pk}$  and  $\text{vk}$  thus remain constant across nodes and epochs, while each proof  $\Pi_t^j$  is unique to its witness. Security guarantees for completeness, soundness, and zero-knowledge follow directly from the Groth16 construction and are summarised in Sec. 5.2.1.

*Remark 1:* ZKPs are generally categorized into two main types: SNARKs (Succinct Non-Interactive Arguments of Knowledge) and STARKs (Scalable Transparent Arguments of Knowledge). In our scheme, we utilize ZK-SNARKs due to their succinct proofs, efficient verification, and

---

<sup>1</sup>A simple hash is sufficient at this stage; Sec. 5.2.1 gives the concrete KZG instantiation and its 35 k-gas verification cost.

compatibility with blockchain-based systems. SNARKs, such as Groth16 [195] and Pinocchio [196], rely on elliptic curve cryptography and a trusted setup, which is effectively managed in our scheme by an oracle. While STARKs offer advantages such as transparency (no trusted setup) and post-quantum security, their larger proof sizes and higher computational costs make them less practical in our context. For a detailed discussion on these ZKP methods, see [197], [198].

*Remark 2:* To prevent proof-generation latency from inflating overall training time, ZK-HybridFL employs a *predict-then-prove* workflow. Immediately after completing a local training step, each node (i) broadcasts its model update together with its hash, and then (ii) proceeds with the next SGD iteration without waiting for the zero-knowledge proof to be produced. Proofs are generated asynchronously in the background and thus are effectively amortized over subsequent training steps. A two-epoch (configurable) grace period is provided before any proof failures trigger the challenge mechanism, ensuring that transient delays do not stall the protocol under normal hardware conditions.

## Cryptographic Instantiation

The generic workflow above abstracts away the precise commitment scheme and the algebraic details of the SNARK. We now spell out the concrete instantiation used in our implementation.

**Polynomial commitments.** Let  $\mathbf{W}_j^t \in \mathbb{F}_p^n$  be the trainer’s weight tensor at round  $t$ , flattened into the length- $n$  vector

$$\mathbf{W}_j^t = (W_{j,0}^t, W_{j,1}^t, \dots, W_{j,n-1}^t),$$

where each  $W_{j,i}^t \in \mathbb{F}_p$  is one scalar model parameter. Similarly, let the test-batch tensor  $D_j^{t,\text{test}} \in \mathbb{F}_p^m$  be flattened as

$$D_j^{t,\text{test}} = (D_{j,0}^{t,\text{test}}, D_{j,1}^{t,\text{test}}, \dots, D_{j,m-1}^{t,\text{test}}),$$

with each  $D_{j,i}^{t,\text{test}} \in \mathbb{F}_p$  denoting one scalar feature or label value in the batch.

We then interpret these as evaluation vectors of degree- $(n - 1)$  and degree- $(m - 1)$  polynomials

in  $\mathbb{F}_p[x]$ :

$$f_W(x) = \sum_{i=0}^{n-1} W_{j,i}^t x^i, \quad f_D(x) = \sum_{i=0}^{m-1} D_{j,i}^{t,\text{test}} x^i, \quad f_W, f_D \in \mathbb{F}_p[x].$$

Under the KZG setup (secret  $\tau \in \mathbb{F}_p$ ), we commit by

$$C_j^{t,\text{model}} = g_1^{f_W(\tau)}, \quad C_j^{t,\text{test}} = g_1^{f_D(\tau)}.$$

Each commitment is a single 48-byte point in  $\mathbb{G}_1$  and is stored on the side-chain together with a Lamport timestamp (See Sec. 5.3.2).

**Trusted setup.** A single Powers-of-Tau ceremony produces (i) the universal structured reference string  $(g_1^{\tau^k})_{k=0}^{k_{\max}}$  and (ii) the circuit-specific proving key  $\text{pk}$  and verification key  $\text{vk}$  for the Groth16 SNARK. The ceremony is executed once at task deployment; afterwards  $\text{pk}$  is distributed off-chain to trainers, whereas  $\text{vk}$  is pinned on-chain.

**Non-interactive proof.** As described in Sec. II-A-2, using  $\text{pk}$ , trainer  $j$  computes a Groth16 proof  $\Pi_j^t$  based on public inputs. Since the Fiat–Shamir heuristic is applied to derive challenges deterministically, no additional interaction or exchange of randomness is required.

**Verification Contract.** The side-chain validator performs:

- (i) Two KZG opening checks (two pairings + one multi-exp; cost  $\approx 34.2$  k gas and 144 B of storage for the two 48 B commitments),
- (ii) One Groth16 verification call on  $(C_j^{t,\text{model}}, C_j^{t,\text{test}}, \mathcal{Y}_j^t, \mathcal{L}_j^t, \Pi_j^t)$  (cost 20–27 k gas; see Table 5.5).

Even with 30 trainers, this, plus the additional 144 B of on-chain storage, remains under 1% of a 2 M gas block, keeping overhead negligible. If all checks succeed, a **ProofOK** event is emitted. Otherwise, the update is rejected and node  $j$ 's stake becomes challengeable.

**Asynchronous epochs.** Immediately after publishing their commitments  $(C_j^{t,\text{model}}, C_j^{t,\text{test}})$ , each trainer resumes the next SGD epoch while proof generation runs concurrently (the “predict-then-prove” schedule). We allow a two-epoch grace window, which is sufficient even for a 76 s GPU proof for MobileNetV2-0.5 on MNIST (each epoch itself takes 24–30 s). Commitments and proofs carry Lamport timestamps to totally order side-chain events, so late proofs simply land in the next round without any global pause (See Sec. 5.3.2).

## Security Guarantees

The combination of KZG commitments and Groth16 delivers the usual *completeness*, *soundness*, and *zero-knowledge* properties. Below we make those guarantees explicit for the concrete operations performed in ZK-HybridFL.

**Model or data substitution.** Let  $C_j^{t,\text{model}}$  and  $C_j^{t,\text{test}}$  be the commitments posted in step (i) of Sec. 5.2.1, and let  $\Pi = \Pi_j^t$  be the proof broadcast in step (iii). Suppose the prover attempts to convince the verifier with an altered model  $\tilde{\mathbf{W}} \neq \mathbf{W}_j^t$  or test batch  $\tilde{D} \neq D_j^{t,\text{test}}$ . Because KZG is *binding*, the pair of polynomials  $(f_{\tilde{\mathbf{W}}}, f_{\tilde{D}})$  cannot both satisfy

$$g_1^{f_{\tilde{\mathbf{W}}}(\tau)} = C_j^{t,\text{model}} \quad \text{and} \quad g_1^{f_{\tilde{D}}(\tau)} = C_j^{t,\text{test}},$$

so at least one KZG-opening check fails and verification returns 0.

**Fabricating outputs or loss.** Suppose instead the prover keeps the committed witness  $(\mathbf{W}_j^t, D_j^{t,\text{test}})$  but replaces the public outputs by  $\tilde{\mathcal{Y}} \neq \mathcal{Y}_j^t$  or  $\tilde{\mathcal{L}} \neq \mathcal{L}_j^t$ . The algebraic relations embedded in the Groth16 circuit bind the full witness  $\mathcal{U}_j^t$  to the declared outputs, so any mismatch violates a circuit constraint and forces the SNARK verifier to return 0.

**Lazy-node detection.** Each bundle  $Z_j^t$  includes the hash  $H(\mathcal{Y}_j^t)$  alongside  $C_j^{t,\text{model}}$  and  $C_j^{t,\text{test}}$ . Because both KZG and  $H$  are collision-resistant, two bundles that replay an *identical* prediction

vector must derive from the same  $(\mathbf{W}, D)$  witness (and thus the same loss). Smart-contract  $S^1$  cross-checks these hashes across the DAG: any node that replays an old model to avoid fresh training is flagged and its update excluded from the parent-selection ranking (cf. Sec. 5.3.1), mitigating “lazy-node” attacks.

**Consistency across forks.** KZG commitments and the hash  $H(\mathcal{Y})$  are stored in the immutable part of the side-chain; therefore every honest replica of the DAG sees *exactly* the same triplet  $(C^{t,\text{model}}, C^{t,\text{test}}, H(\mathcal{Y}))$ . This global consistency enables (i) deterministic parent selection and (ii) unambiguous fork resolution, even under short-lived network forks.

Taken together, these properties ensure that only *genuinely trained, fully verified* updates influence the global model, while any adversarial attempt, whether by parameter substitution, fabricated outputs, or stale replay is detected and rejected except with negligible probability.

## 5.2.2 DAG

By using a modified IOTA Coordicide [95] consensus mechanism as described next, our proposed scheme employs a DAG ledger to facilitate interactions among trainers nodes while keeping the overall network scalable and secure.

The basic DAG operations used in ZK-HybridFL, such as tip-based parent selection, weight accumulation for confirmation, and graph reachability checks that support the challenge process, are adopted almost verbatim from the Coordicide design of IOTA ledger.

Our contributions lie in **how these established primitives are repurposed for FL**. First, every block must include a zero-knowledge proof that is *verified on-chain* before the block can gain any weight. This guarantees that only correctly trained updates can influence the ledger. Second, the **loss-aware aggregation policy** (Sec. III-A) admits only the most recent *confirmed* blocks and ranks candidate parents by their validated loss. As a result, low-quality or adversarial updates accrue insufficient weight and are naturally pruned from the ledger. To the best of our knowledge, no prior DAG-based blockchain combines proof-gated admission with loss-aware parent selection in the

context of a decentralized parameter server. This dual gating mechanism is the core algorithmic contribution of ZK-HybridFL’s DAG layer.

In our proposed scheme, each node  $j$  submits a block  $D_j(t) = [\mathbf{W}_j^t, Z_j^t]$  to the DAG ledger during epoch  $t$ , where  $\mathbf{W}_j^t$  is the updated local model and  $Z_j^t$  is the ZKP bundle described in Sec. 5.2.1. Blocks  $D_j(t)$  within the DAG can exist in one of three states: confirmed, tip, or unconfirmed. Confirmed blocks ( $D_j^C(t)$ ) have an Aggregated Weight (AW) exceeding a predefined threshold, making them validated and integrated into the stable, immutable part of the DAG. Tip blocks ( $D_j^T(t)$ ) are the most recent blocks at the DAG’s frontier, ready to be extended by incoming blocks. Unconfirmed blocks ( $D_j^U(t)$ ) are those added to the DAG but still in a pending state, awaiting sufficient AW to reach confirmation.

During each epoch  $t$ , the global model  $\tilde{\mathbf{W}}_j^t$  is obtained through local model aggregation using the model updates from the most recently confirmed blocks (see Sec. 5.3.1). Next, we provide a detailed explanation of the consensus mechanism for the DAG component and the computation of the AW used for block confirmation within the DAG.

To submit its contributed block  $D_j(t)$  for epoch  $t$  to the DAG, node  $j$  begins by randomly selecting  $K_T$  tip blocks. These chosen blocks form a set  $\mathcal{D}_j^T(t)$  of potential parents with  $|\mathcal{D}_j^T(t)| = K_T$ . For each block  $D_{j'}^T(t')$  in this set, where  $t' < t$ , node  $j$  verifies its ZKP and extracts its associated loss. As a result, node  $j$  identifies  $K_V$  blocks with valid ZKPs that have the lowest loss values to form the parent block set  $\mathcal{D}_j^V(t)$  for  $D_j(t)$ , with  $|\mathcal{D}_j^V(t)| = K_V$ . Then  $D_j(t)$  is labeled as a tip block  $D_j^T(t)$ , and all tip blocks  $D_{j'}^T(t') \in \mathcal{D}_j^V(t)$  are changed to unconfirmed status, i.e.,  $D_{j'}^U(t')$ . Finally,  $D_j^T(t)$  is integrated into the DAG by selecting the blocks in  $\mathcal{D}_j^V(t)$  as its predecessors. As a result of above actions performed by all nodes, the AW values of the blocks on the DAG will change, which leads to a change in the validity status of some blocks on the DAG. Specifically, certain blocks  $D_{j''}^U(t'')$  for  $t'' < t' < t$ , become confirmed, i.e., they are updated to  $D_{j''}^C(t'')$ . Next, we explain how the AW of each block on the DAG is altered and how the transition from a tip to a confirmed status occurs.

Fig. 5.1 illustrates a DAG ledger for two consecutive epochs  $t - 1$  and  $t$ . In epoch  $t - 1$ , each vertex represents a block  $D_{j_i}(t_i)$  by node  $j_i$ , for  $i = 1, \dots, 4$ , at epoch  $t_i < t - 1$ . A direct edge from

one vertex to another, for example, from  $D_{j_2}^T(t_2)$  to  $D_{j_1}^U(t_1)$ , signifies that  $j_2$  verifies block  $D_{j_1}^U(t_1)$ . Fig. 5.1 also displays the status evolution of the blocks after one epoch. A block is labeled as a *tip* if it lacks incoming validation edges. Therefore, in this figure, for epochs  $t - 1$  and  $t$ , the blocks  $\{D_{j_2}^T(t_2), D_{j_3}^T(t_3), D_{j_4}^T(t_4)\}$  and  $\{D_{j_5}^T(t_5), D_{j_6}^T(t_6)\}$  are tip blocks, respectively.

Moreover, a block that is neither a tip nor confirmed is referred to as an unconfirmed block. Hence, while blocks  $\{D_{j_2}^T(t_2), D_{j_3}^T(t_3), D_{j_4}^T(t_4)\}$  are tips in epoch  $t - 1$ , block  $D_{j_3}^U(t_3)$  is classified as unconfirmed in epoch  $t$ , while  $\{D_{j_2}^T(t_2), D_{j_4}^T(t_4)\}$  remain as tips. Observe that the status of  $D_{j_1}^U(t_1)$ , transitions from unconfirmed to confirmed, which is denoted by  $D_{j_1}^C(t_1)$ . Specifically, a block achieves confirmed status when its AW surpasses a predefined threshold  $\eta$ . The AW is intricately linked to a weight vector that reflects the relative influence of the nodes.

The weight  $\omega_j$  of each node  $j$  with  $\omega_j > 0$ ,  $\sum_j \omega_j = 1$ , is determined by the number of tokens it has staked. In DAG ledgers, staking refers to the process by which nodes lock their tokens to gain voting power in validating transactions and appending new blocks to the ledger. The more tokens a node stakes, the higher its associated validation weight, influencing the cumulative weight of the blocks it approves. This weight is assumed to remain constant over different epochs, but can be dynamically adjusted based on node behavior. Specifically, if a node incorrectly disputes the validity of a block as discussed in the challenge mechanism outlined in Sec. 5.2.3, then a portion of its staked tokens is slashed as a penalty, directly reducing its weight  $\omega_j$ . This incentivizes honest participation by making malicious behavior costly. To establish the network's initial trust and ensure a secure starting point, a set of genesis blocks with *confirmed* status are proposed by oracle (as discussed in Sec. IV-B2) as trusted and correct blocks, providing a foundation for subsequent validations.

The AW of a block  $D_{j_i}(t_i)$  is calculated as  $\omega_i + \sum_{D_{j_{i'}}(t_{i'}) \in \mathcal{F}(D_{j_i}(t_i))} \omega_{j_{i'}}$ , where  $\mathcal{F}(D_{j_i}(t_i))$  denotes the future cone of  $D_{j_i}(t_i)$ , encompassing all blocks that it validates, either directly or through a series of validations. For example, in Fig. 5.1 and in epoch  $t$ , blocks  $\{D_{j_1}^U(t_1), Z_{j_4}^U(t_4)\}$  are in the future cone of block  $D_{j_6}^T(t_6)$ , as block  $D_{j_4}^U(t_4)$  is validated by it directly, while block  $D_{j_6}^T(t_6)$  validates block  $D_{j_1}^C(t_1)$  indirectly. Moreover, Fig. 5.1 illustrates that the AW of block  $Z_{j_1}(t_1)$  surpasses  $\eta$  after the addition of the weights of blocks  $D_{j_5}^T(t_5)$  and  $D_{j_6}^T(t_6)$  and its status transits

from an unconfirmed block to confirmed

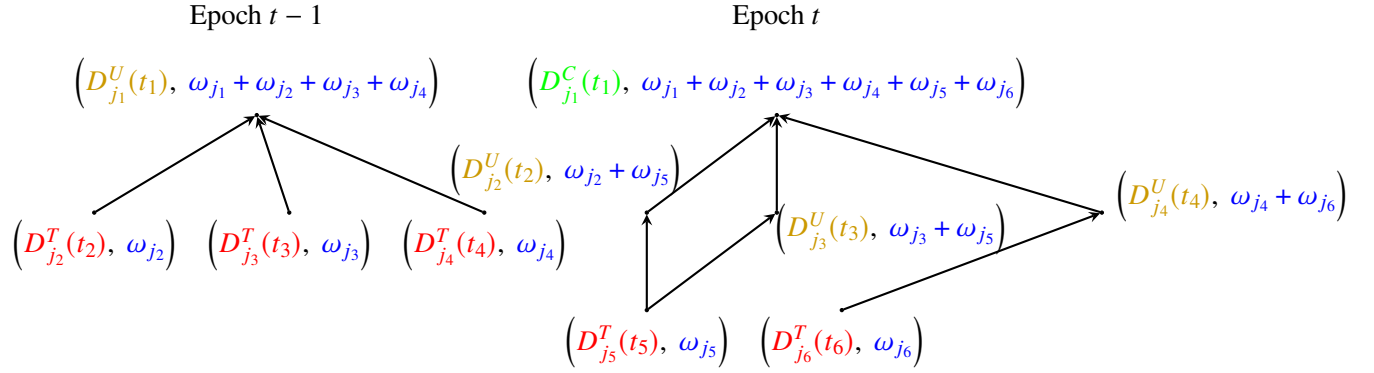


Figure 5.1: Evolution of a DAG ledger over two consecutive epochs. Nodes are color-coded to indicate their status: red for tips, yellow for unconfirmed, and green for confirmed. The blue text represents the AW associated with each node.

### 5.2.3 Challenge Mechanism

In DAG ledgers, each block submission requires selecting parent blocks from the current set of tip blocks. Under normal circumstances, this process ensures that new blocks are seamlessly integrated into the network by referencing recent, valid blocks. However, a critical vulnerability known as the *orphanage attack* [109] exploits this mechanism. In an orphanage attack, malicious nodes intentionally select their own invalid blocks, or those of their colluding partners, as parents when submitting new blocks. By continuously referencing their own invalid blocks, these dishonest nodes effectively remove such blocks from the tip pool. Consequently, other honest nodes are unable to examine the validity of these blocks while selecting parents, allowing the invalidity of these strategically added blocks to go unnoticed. As a result, these invalid blocks remain hidden within the DAG despite being removed from the tip set. Later on, as the DAG grows, valid blocks may indirectly reference these previously isolated invalid blocks, causing them to accumulate weight over time. Ultimately, this process can lead to the confirmation of invalid blocks. The confirmation of such blocks not only compromises the quality of the global model derived from confirmed blocks but also alters the topology of the network, disrupting the integrity of the ledger.

The challenge mechanism is motivated by the need to counter this type of attack and restore

transparency and fairness to the block selection process. This mechanism empowers honest nodes to flag and contest blocks that remain in an *unconfirmed* status within the DAG. The process begins with the detection of blocks impacted by an orphanage attack. At the core of this detection mechanism is the Graph Reachability Analysis (GRA) [199]–[201], a structural algorithm designed to identify blocks that have caused other blocks to become isolated within the DAG. The key principle behind GRA is that any valid block should be reachable from at least one tip block through a path in the DAG graph. If a block is found to be disconnected from all active tips, it indicates a potential issue: either the block was unintentionally abandoned due to network delays, or it was strategically isolated as part of an orphanage attack.

For each suspicious block identified through GRA, the corresponding ZKP is verified. By definition, blocks involved in an orphanage attack are invalid, and their associated ZKPs will fail verification. Once such blocks are detected, the detecting node broadcasts a challenge to all trusted oracles within the network. Upon receiving a challenge, the oracles compile a list of disputed blocks and independently verify their validity based on the provided ZKPs.

Following verification, the oracles participate in a supermajority voting process: If more than  $2/3$  of the oracles agree that a block is invalid, it is formally revoked from the DAG. Block revocation means that the invalid block is excluded from future parent selections, and the AW of affected blocks is updated accordingly to neutralize the impact of the revoked block on the network. Conversely, if a supermajority of oracles determine that the disputed block is valid, the network penalizes the challenging node  $j$  by slashing a fraction of its tokens. This penalty reduces the challenger's  $\omega_j$ , thereby decreasing its influence and voting power within the DAG for future block submissions.

*Remark 3:* While a wrong dispute results in penalizing the disputer node, ZK-HybridFL does not penalize the submitter of an invalid block. This design choice reflects a balance between encouraging participation and acknowledging that honest mistakes or network delays can lead to inadvertent errors. Penalizing such actions could stifle honest contributions, especially when no malicious intent is involved. In contrast, initiating a challenge, especially one that incorrectly disputes a valid block, incurs significant costs by triggering a consensus process among oracles resulting in computational

overhead.

#### 5.2.4 Extended ZKP Defenses

##### **Broader Threat Model**

While the core workflow of Sec. II-A ensures that blatantly bogus updates (e.g., completely untrained models, fabricated outputs, or exact replays) are caught with overwhelming probability, a realistic threat model must also consider subtler strategies that deliberately skirt these checks. Below, we sketch three attack avenues that slip through the ZKP workflow in Sec. II-A.

First, a node skips training altogether and re-posts an older weight tensor  $\mathbf{W}_j^{t-\Delta}$  with a microscopic perturbation. This changes the hash of the predictions, so replay detection in Sec. II-A does not trigger, but the global model sees no genuine progress.

Second, instead of performing full local SGD, the node applies a single gradient step  $\delta\mathbf{W}$  and then scales it by a tiny factor  $\alpha$ . The resulting update  $\mathbf{W}_j^{t-1} + \alpha\delta\mathbf{W}$  meets the norm bound and earns the block reward, yet makes negligible impact on the model’s decision surface.

Third, since the test batch  $D_j^{t,\text{test}}$  is chosen privately, a malicious trainer can cherry-pick samples on which  $\mathbf{W}_j^t$  already performs well, or even craft an easy synthetic set, then prove the accuracy honestly. The ZKP verifies, but the reported loss overstates the model’s real-world quality.

To address these security gaps, next we introduce lightweight extensions that detect and penalize the three strategies above, without reshaping the Groth16-based core.

##### **Extended ZKP Bundle**

In each training epoch  $t$ , we extend the original “ZKP-bundle” from Sec. II-A with two lightweight, non-interactive proofs to ensure that every submitted update both moves the weights by a meaningful amount and alters the model’s internal representations. The efficiency of these methods has been demonstrated in [202], [203]. We defer a detailed security analysis of these extended ZKPs to Sec. 5.5.7.

Specifically, in epoch  $t$ , each client  $j$  is given the public thresholds  $[L_t, B_t]$  and  $\tau_{\max}$ . See

Sec. III-C for how these thresholds are computed and how consensus is reached across the network through interactions with the oracle committee, sidechain, and smart contracts.

With these parameters in hand, each node  $j$  proceeds as follows in epoch  $t$ :

**1. Constructs the original Z-bundle.**

The node first performs standard local training and generates a Groth16 proof  $\Pi_j^t$  verifying inference correctness. It then commits the model and test data and assembles the initial Z-bundle:

$$Z_j^t = \left( \Pi_j^t, \mathcal{Y}_j^t, \mathcal{L}_j^t, C_j^{t,\text{model}}, C_j^{t,\text{test}} \right)$$

**2. Proves the norm of the update with Bulletproofs.**

To ensure that the weight change is non-trivial, the node computes the update  $\Delta W_j^t = W_j^t - W_j^{t-1}$  and produces a *range*-proof, i.e., Bulletproof:

$$\Sigma_j^t = \text{BPProve}(\Delta W_j^t, L_t, B_t)$$

This proof references the existing commitment  $C_j^{t,\text{model}}$  to link the norm constraint to the committed model.

**3. Attests to semantic change using embedding-cosine SNARK.**

The node computes layer- $\ell$  activations over a public probe set:

$$z_{\text{old}} = \frac{1}{|D_{\text{probe}}|} \sum_{x \in D_{\text{probe}}} \phi_\ell(W_j^{t-1}, x),$$

$$z_{\text{new}} = \frac{1}{|D_{\text{probe}}|} \sum_{x \in D_{\text{probe}}} \phi_\ell(W_j^t, x),$$

and generates a SNARK proving  $\cos(z_{\text{old}}, z_{\text{new}}) \leq \tau_{\text{max}}$ :

$$\Gamma_j^t = \text{SNARKProve}(\cos(z_{\text{old}}, z_{\text{new}}) \leq \tau_{\text{max}} ; C_{\text{probe}})$$

#### 4. Assembles the extended bundle.

The node merges the original bundle with the two additional proofs:

$$\begin{aligned} Z_j^{t,\text{ext}} &= Z_j^t \parallel \Sigma_j^t \parallel \Gamma_j^t \\ &= \left( \Pi_j^t, \Sigma_j^t, \Gamma_j^t, \mathcal{Y}_j^t, \mathcal{L}_j^t, C_j^{t,\text{model}}, C_j^{t,\text{test}} \right) \end{aligned}$$

#### 5. Submits for peer or on-chain verification.

A verifier node  $j'$  or the chain checks all parts in order:

- (a)  $\text{VerifyGroth16}(vk, \Pi_j^t, C_j^{t,\text{model}}, C_j^{t,\text{test}}, \mathcal{Y}_j^t, \mathcal{L}_j^t)$
- (b)  $\text{VerifyBulletproof}(\Sigma_j^t; C_j^{t,\text{model}})$
- (c)  $\text{VerifyCosineZKP}(\Gamma_j^t; C_{\text{probe}})$

The update is accepted into the DAG only if all verifications succeed.

Also, note that  $\Sigma_j^t$  must reference  $C_j^{t,\text{model}}$  so that the norm bound applies to the committed weights, and  $\Gamma_j^t$  refers to the fixed public probe-set commitment  $C_{\text{probe}}$ . No additional commitments are required.

Moreover, in addition to the two KZG openings (approximately 34.2 k gas) and the Groth16 verification (20–27 k gas), each update in the extended scheme carries:

- A Bulletproof  $\Sigma_j^t$  for the  $\ell_2$ -bound (proof size  $\approx 8.1$  kB off-chain), which costs  $\approx 38$  k gas to verify.
- An embedding-cosine SNARK  $\Gamma_j^t$  (proof size  $\approx 200$  B), which costs  $\approx 25$  k gas to verify.

Taken together, the two new verifications add roughly 63 k gas, under 3.2% of a 2 M-gas block, on top of the original  $\approx 61$  k, keeping total overhead below 5% of block capacity. See Table 5.6 for a full breakdown.

*Remark 4:* Although we have extended the original Z-bundle, the underlying DAG ledger and its consensus rules (Sec. II-B) remain unchanged. Likewise, the challenge mechanism of Sec. II-C applies verbatim to each extended bundle: any node may still issue and resolve challenges against  $Z_j^{t,\text{ext}}$  under the same stake-based rules.

### 5.3 EDSC Mechanism of ZK-HybridFL

Our proposed ZK-HybridFL system employs Event-Driven Smart Contracts (EDSCs) to automate predefined actions in response to specific network events. By routing every off-chain emission through a single `EventAdmission` contract (backed by the `OracleRegistry`), we avoid constant polling and heavyweight sidechain consensus, while ensuring that only vetted, canonical events can trigger on-chain logic. Sec. IV.A details the main EDSCs used in ZK-HybridFL, and Sec. 5.3.2 describes the Oracle Committee’s verification and event-publication process.

In ZK-HybridFL, all EDSCs are deployed on a dedicated sidechain that isolates contract execution from the main ledger. This design boosts throughput and scalability by handling high-frequency interactions off-chain, without burdening the primary network. Sec. 5.3.2 presents the sidechain architecture and its integration within the broader protocol.

#### 5.3.1 Smart Contracts in ZK-HybridFL

Each node  $j$  in the ZK-HybridFL process deploys five EDSCs  $\{S_j^k\}_{k=1}^5$  on the sidechain, which are independently triggered by validated events by the oracle. Next we delineate the function and process of each of these five EDSCs.

##### **Validation Smart Contract $S_j^1$**

$S_j^1$  validates the ZK bundles of the blocks in  $\mathcal{D}_j^T(t)$ . It subscribes to the `EventApproved` log emitted by the `EventAdmission` contract (cf. Section 5.3.2), which the Oracle Committee emits once it has vetted node  $j'$ 's DAG-published bundle  $Z_{j'}^t$ . For each block  $D_{j'}^T(t') \in \mathcal{D}_j^T(t)$ ,  $S_j^1$  retrieves from the side-chain the commitments  $C_{j'}^{t',\text{model}}$ ,  $C_{j'}^{t',\text{test}}$ , and the global verification key  $\text{vk}$ . It then fetches  $\Pi_{j'}^t$ ,  $\mathcal{Y}_{j'}^t$ , and  $\mathcal{L}_{j'}^t$  from the bundle  $Z_{j'}^t$  on the DAG, invokes `Verify`( $\text{vk}$ ,  $\Pi_{j'}^t$ ,  $C_{j'}^{t',\text{model}}$ ,  $C_{j'}^{t',\text{test}}$ ,  $\mathcal{Y}_{j'}^t$ ,  $\mathcal{L}_{j'}^t$ ), and retains the block only if the call returns 1. After processing all candidates,  $S_j^1$  ranks the verified blocks by their loss values  $\mathcal{L}_{j'}^t$  and selects the top  $K_V$  with the lowest loss to form  $\mathcal{D}_j^V(t)$ . This finalizes the contract’s state, enabling node  $j$  to adopt these blocks as parents in the next stage of

ZK-HybridFL.

### **Submission Smart Contract $S_j^2$**

$S_j^2$  facilitates the submission of node  $j$ 's new block  $D_j(t)$  to the DAG and updates the AW of existing blocks. It subscribes to the `EventApproved` log emitted by the `EventAdmission` contract (cf. Section 5.3.2), which is triggered once the Oracle Committee has vetted and approved the validated set  $\mathcal{D}_j^V(t)$ . The event communicates that node  $j$  has determined the parent blocks from  $\mathcal{D}_j^V(t)$  for its new block  $D_j(t)$ ; it includes necessary information on these parent blocks alongside the model update  $\mathbf{W}_j^t$  and corresponding ZKP  $Z_j^t$  contained in  $D_j(t)$ . It then updates the AW values associated with the vertices corresponding to these blocks within  $\mathcal{D}_j^V(t)$  on the DAG and their ancestors, acknowledging their role as parents of the new block. Following this update,  $S_j^2$  labels  $D_j(t)$  as a new *tip* of the DAG (denoted  $D_j^T(t)$ ), while altering the status of each parent block  $D_{j'}^T(t') \in \mathcal{D}_j^V(t)$  from *tip* to *unconfirmed* (denoted  $D_{j'}^U(t')$ ). If the updated AW of any block exceeds a predetermined threshold,  $S_j^2$  marks such blocks as *confirmed*. In its final step,  $S_j^2$  transitions its state to complete by embedding  $D_j^T(t)$  into the DAG as a new vertex. This vertex is connected via directed edges to the vertices corresponding to the parent blocks in  $\mathcal{D}_j^V(t)$ , effectively updating the DAG structure to reflect the addition of node  $j$ 's block and the revised parent relationships.

### **Challenge Smart Contract $S_j^3$**

$S_j^3$  manages the dispute mechanism for potentially invalid blocks within the ZK-HybridFL framework. It subscribes to the `EventApproved` log emitted by the `EventAdmission` contract (cf. Section 5.3.2), which the Oracle Committee emits once it has vetted node  $j$ 's off-chain GRA detection of a suspicious block  $D_{j'}(t')$ . Upon triggering,  $S_j^3$  receives the GRA detection results as inputs comprising the identifiers of the suspicious blocks  $\{D_{j'}(t')\}$ , the identity of the node  $j'$  that created them, relevant ZKPs or metadata supporting the suspicion, and a fraction of node  $j$ 's tokens staked as collateral.

Following receipt of these inputs,  $S_j^3$  notifies trusted oracles of the questionable blocks by

emitting an event containing details of the suspicious blocks, GRA findings, and supporting evidence. Simultaneously, the contract holds the staked tokens in escrow. This notification prompts the oracles to validate the challenge: they assess the evidence to determine the validity of the suspicious blocks. The output of  $S_j^3$  thus involves alerting the oracles and securely holding the stake while awaiting their consensus. Depending on the oracles' decision,  $S_j^3$  then updates its state accordingly. If the oracles confirm that the challenged blocks are invalid,  $S_j^3$  marks these blocks as revoked within the DAG ledger, ensuring the integrity of the ledger by potentially reconnecting or adjusting references as needed, and releases the staked tokens back to node  $j$ . Conversely, if the oracles find the challenge unfounded,  $S_j^3$  imposes a penalty by slashing the staked tokens, reducing node  $j$ 's future influence, and maintains the status of the challenged blocks. This final state transition solidifies the resolution of the dispute, either by cleansing the DAG of invalid blocks or by penalizing incorrect challenges.

#### **Model Aggregator Smart Contract $S_j^4$**

$S_j^4$  is assigned to compute the global model  $\tilde{\mathbf{W}}_j^t$  for node  $j$ . It subscribes to the `EventApproved` log emitted by the `EventAdmission` contract (cf. Section 5.3.2), which the Oracle Committee emits once node  $j$  has signaled availability for epoch  $t + 1$  and the dispute resolution in  $S_j^3$  has concluded. This event carries the latest AW of DAG blocks along with their updated validity status. Upon receiving that log,  $S_j^4$  determines the set  $\mathcal{J}_j^t$  of all nodes  $j'$  that produced at least one new confirmed block in epoch  $t$ . For each  $j'$ , it identifies the most recent epoch  $t_{j'}^*$ , such that  $D_{j'}^{t_{j'}^*}$  became confirmed in epoch  $t$ , and retrieves the corresponding model update  $\mathbf{W}_{j'}^{t_{j'}^*}$ . The contract then computes the global model via weighted aggregation:

$$\tilde{\mathbf{W}}_j^t = \sum_{j' \in \mathcal{J}_j^t} \frac{\omega_{j'}}{\sum_{k \in \mathcal{J}_j^t} \omega_k} \mathbf{W}_{j'}^{t_{j'}^*}, \quad (5.1)$$

ensuring that only the latest confirmed update from each node contributes and that weights are normalized over  $\mathcal{J}_j^t$ . Once aggregation completes,  $S_j^4$  transitions to the complete state and returns  $\tilde{\mathbf{W}}_j^t$  to node  $j$ , enabling it to commence the next training epoch.

## Reward distribution Smart Contract $S_j^5$

$S_j^5$  is allocated for dispensing rewards to trainer node  $j$  upon successful submission of its contributed block  $D_j^T(t)$ . It subscribes to the `EventApproved` log emitted by the `EventAdmission` contract (cf. Section 5.3.2), which the Oracle Committee emits once the submission process in  $S_j^2$  has concluded. This event provides information regarding the contributed block  $D_j^T(t)$ , its corresponding model  $\mathbf{W}_j^t$ , and the verification details confirming successful integration into the DAG. Specifically, once triggered,  $S_j^5$  verifies the final status of block  $D_j^T(t)$  and the correctness of  $\mathbf{W}_j^t$ , as endorsed by the oracle. It then calculates the reward amount based on established criteria such as model accuracy and the number of valid blocks that node  $j$  has successfully introduced into the system. After finalizing the reward,  $S_j^5$  transitions its state to complete by creating a transaction on the sidechain to credit the trainer node  $j$ 's account with the calculated sum.

Table 5.1 summarizes the structure of the smart contracts for ZK-HybridFL where the input, output, and state variables of these EDSCs in epoch  $t$  are denoted as  $\{I_j^k(t), O_j^k(t), P_j^k(t)\}_{k=1}^5$ .

Table 5.1: Summary of EDSCs for ZK-HybridFL.

SC	Major Defined Event Subscription	Input	Output	State
$S_j^1$	Generation of ZKP $Z_j^t$	Triggers validation of proofs $Z_j^t$ in blocks $D_j^T(t')$	Validation results of ZKPs and performance evaluations, forming set $\mathcal{D}_j^V(t)$	Transition to complete after sending results to node $j$
$S_j^2$	Formation of set $\mathcal{D}_j^V(t)$	Identifies and verifies valid blocks within $\mathcal{D}_j^V(t)$ on the DAG	Updates DAG by labeling block $D_j(t)$ as tip $D_j^T(t)$ , changes status of parent blocks	Releases contributed block $D_j^T(t)$ into the DAG
$S_j^3$	Detection of potentially invalid block $D_{j'}(t')$	Receives GRA detection results and token stake from node $j$ , including flagged blocks $\{D_{j'}(t')\}$ and relevant ZKPs	Notifies oracles of suspicious blocks and stakes tokens in escrow; oracles verify challenge validity and determine slashing or refund	Updates DAG by marking invalid blocks and slashing tokens if the challenge fails; returns tokens if challenge succeeds
$S_j^4$	Announcement of trainer availability for next epoch $t + 1$	Aggregates model updates $\mathbf{W}_{j'}^t$ from confirmed blocks $D_{j'}^C(t')$	Generates global model $\tilde{\mathbf{W}}_j^t$ for epoch $t + 1$ via aggregation rule	Sends global model $\tilde{\mathbf{W}}_j^t$ to node $j$
$S_j^5$	Formation of block $D_j^T(t)$	Verifies completion and integration of model update $D_j^T(t)$ and model $\mathbf{W}_j^t$	Determines reward amount based on predefined criteria (e.g., accuracy, contribution quality)	Releases calculated reward to trainer node $j$ via blockchain transaction

### 5.3.2 Oracle-assisted Sidechain

#### Oracle Committee: Event-Admission Layer

ZK-HybridFL introduces a lightweight, fault-tolerant Oracle Committee that serves as an event-admission layer between the DAG and the Event-Driven Smart Contracts (EDSCs). Rather

than allowing nodes to invoke contracts directly, which means risking malformed or out-of-context messages, the committee observes each raw DAG emission, applies a suite of structural and contextual checks, and only publishes a succinct, threshold-signed `EventApproved` log on-chain. By decoupling schema and consistency validation from on-chain business logic, we ensure that EDSCs can react immediately and safely, without incurring bulky consensus or verification costs.

Membership in the Oracle Committee aligns with the same Byzantine-fault threshold as the underlying DAG. We require  $M \geq 3f + 1$  full nodes to register as oracles, where  $f$  is the maximum number of Byzantine failures tolerated. Each candidate deposits collateral and publishes a BLS public key in an on-chain `OracleRegistry`; only active, staked nodes participate in validation. Should a member prove adversarial or offline for too many consecutive events, an EDSC-driven slashing mechanism automatically revokes its status and redistributes its stake. This registry not only governs entrance and exit but also encodes the threshold  $f + 1$  necessary for event publication.

Once the committee is formed, each oracle runs an off-chain watcher that listens to the DAG for new raw events  $e = \{\text{type}, \text{payload}, \text{meta}\}$ . Upon observing  $e$ , the node verifies conformity to the expected schema, cross-references any referenced block hashes or commitments against the DAG's current state, checks stake requirements or performance metrics, and ensures that Lamport timestamps advance monotonically. If all checks succeed, the node computes a BLS partial signature  $\sigma_i$  on the hash  $h = H(e)$  and gossips  $(e, \sigma_i)$  to its peers. This approach leverages off-chain computation, which is two orders of magnitude faster than on-chain proof verification, while preserving auditability.

When any oracle collects  $f + 1$  valid partial signatures on  $h$ , it combines them into a single BLS threshold signature  $\Sigma$  and submits one compact transaction to the `EventAdmission` contract. That transaction does nothing more than emit the standardized log

```
// EventAdmission.sol
event EventApproved(bytes32 indexed hash, bytes payload);
```

This Solidity snippet represents the canonical interface for approved event emission.<sup>2</sup>

---

<sup>2</sup>The Solidity snippets in this section serve as protocol-level specifications of contract interfaces and behaviors. In

where `hash = h` and `payload` encodes the original event’s essential fields. By limiting on-chain work to a single BLS-verify and log emission, we keep gas costs and block congestion to a minimum: the entire approval process amounts to a few dozen thousand gas, reimbursed from the network’s fee pool. Crucially, any light client or side-chain participant can independently re-compute  $h = H(e)$  and run a local BLS verification against the committee’s public key to confirm both the integrity of the event and the quorum that approved it.

Once the `EventApproved` log appears in the side-chain, all Event-Driven Smart Contracts that have been coded to “subscribe” to this signature automatically wake up. In Solidity this takes the form of a handler such as

```
function onEventApproved(bytes32 hash, bytes calldata payload) external {
    require(msg.sender == address(EventAdmission));
    // decode and enforce domain rules, e.g. stake thresholds, performance
}
```

Each contract thus carries its own domain-specific checks whether labeling a new DAG tip, disbursing rewards, slashing misbehaving participants, or aggregating model updates confident that malformed or replayed messages cannot slip through. The separation of concerns ensures that off-chain vetting remains focused on message consistency, while on-chain logic governs economic and security policies without redundant validation overhead.

To support evolving network conditions and to guard against long-term stagnation or collusion, committee membership is fully updatable. Prospective nodes stake collateral and call `join()` on the `OracleRegistry`, while existing members can be slashed via dedicated dispute contracts for misbehavior or liveness failures. When a membership rotation is desired, a new registry is deployed, and during a hand-off period both old and new sets co-sign events for  $k$  epochs, ensuring no gap in approval coverage. By encoding the registry’s address in each EDSC (via an updatable pointer),

---

our simulation setup, these contracts are implemented as WebAssembly modules using Substrate’s `ink!` smart contract framework. The logic, event signatures, and access controls are preserved identically.

committees can be refreshed without redeploying business-logic contracts, preserving continuity and decentralization over the network’s lifetime.

On commodity hardware the only on-chain work performed by the `EventAdmission` contract is a single BLS verification and log emission. Empirical micro-benchmarks show that aggregating and verifying a threshold BLS signature for  $f + 1$  parties requires approximately 1.4 ms of CPU time, while publishing the transaction consumes roughly 35 k gas and carries a payload under 512 bytes. Even in a worst-case training scenario with 32 approved events per epoch, total off-chain CPU overhead remains below 0.1 s and network bandwidth under 35 kB, and the on-chain gas cost accounts for less than 1% of a 2 M-gas block.

Table 5.2: Oracle Committee micro-benchmarks (per approved event).

Operation	Latency	Gas / Data
Threshold BLS verify ( $f + 1$ )	1.4 ms	96 B signature
<code>publishApprovedEvent</code> TX	–	35 k gas, 512 B payload

That said, the Oracle Committee functions purely as a *Consistency Gate*, isolating structural and contextual checks from the contracts themselves. By emitting a single, threshold-signed `EventApproved` log, it guarantees that downstream EDSCs only ever process canonical, well-formed events. All substantive business logic including tip selection, reward distribution, slashing, model aggregation, remains inside the smart contracts, which trust but verify the committee’s work. This design preserves the DAG’s security assumptions, enables seamless committee rotation, and minimizes both on-chain and off-chain overhead.

## Sidechain

The ZK-HybridFL sidechain serves as a specialized ledger for storing and executing EDSCs, ensuring that high-frequency interactions do not overload the DAG. Its block structure is designed to encapsulate the data elements essential for the protocol’s operation and network logistics. In the initial blocks, the sidechain records identifiers for the members of the Oracle Committee (as registered in the `OracleRegistry`), along with stake-related metadata determining each node’s

weight  $\omega_j$ . During the network’s one-time deployment phase, EDSCs are deployed on the sidechain. Once live, subsequent sidechain blocks record only approved events, i.e., the ‘EventApproved’ logs emitted by the EventAdmission contract pinning the universal SNARK keys  $\text{pk}, \text{vk}$  and capturing dynamic data such as KZG commitments  $C_j^{t,\text{model}}, C_j^{t,\text{test}}$ , reward distributions for successful model submissions, and updates to participants’ staked tokens.

Unlike conventional blockchains that run full consensus (e.g. PoW) on every block, the ZK-HybridFL sidechain leverages its event-driven architecture in lieu of a separate consensus protocol. Whenever ‘EventAdmission’ emits an ‘EventApproved’ log, the sidechain simply attaches the corresponding block with the timestamp of that transaction; blocks are ordered by their inclusion time rather than by a consensus-based proposal process. This built-in ordering mechanism eliminates the need for additional consensus overhead, simplifying transaction validation and ordering on the sidechain.

Relying on physical clocks across a decentralized network introduces challenges like clock drift and latency [204]. To avoid these, ZK-HybridFL uses Lamport logical clocks [205], carried in each approved event’s metadata. Oracle Committee members increment local counters and propagate logical timestamps alongside ‘EventApproved’ logs, yielding a consistent causal ordering without a trusted time authority. This ensures conflict-free execution of EDSCs and state reconciliation across all nodes without traditional consensus protocols. For further details on Lamport-clock ordering in decentralized systems, see [206].

### 5.3.3 Adaptations for Extended ZKP

In this subsection we describe the minimal extensions to our side-chain, Oracle Committee workflow, and smart contract logic required to support the two new proofs ( $\Sigma_j^t$  and  $\Gamma_j^t$ ) in the extended ZKP. We first explain how the committee computes and publishes the per-epoch thresholds  $[L_t, B_t]$  and  $\tau_{\text{max}}$ , then detail the single contract update needed to consume them, and finally summarize the unchanged components.

## Threshold Computation and Dissemination

At the close of epoch  $t - 1$ , each Oracle Committee member gathers the public inputs from all confirmed bundles in the submitted blocks on DAG, specifically, the  $\ell_2$ -norms  $\|\Delta W_k^{t-1}\|$  and cosine-similarity scores on the fixed probe set. By taking the median of the norms and scaling it via committee-chosen factors  $r$  and  $\rho$ , they fix

$$B_t = r \cdot \text{median}\{\|\Delta W_k^{t-1}\|\}, \quad L_t = \rho B_t, \quad 0 < \rho < 1 < r.$$

Concurrently, the 95th percentile of the published similarity scores  $s_k^{t-1} = \cos(z_{\text{old}}, z_{\text{new}})$  on  $D_{\text{probe}}$  (a small public probe set of, e.g., 300–500 samples that could be periodically re-sampled every several epochs by the committee) is chosen as  $\tau_{\text{max}}$ . Because  $D_{\text{probe}}$  is fixed and public, these embeddings cannot be tailored by clients, ensuring  $\tau_{\text{max}}$  reflects genuine semantic shifts.

Finally, the committee emits threshold-signed events `NormThresholdsPublished(epoch,t,L_t,B_t)` and `CosineThresholdsPublished(epoch,t,_max)` via the existing `EventAdmission` contract. All side-chain validators and Event-Driven Smart Contracts subscribe to these `EventApproved` logs and cache  $[L_t, B_t, \tau_{\text{max}}]$  for use in subsequent proof verifications.

## Extension of the Validation Contract $S_j^1$

The only smart contract that requires modification to accommodate the extended  $Z$ -bundle is the per-node validation contract  $S_j^1$ . As before,  $S_j^1$  subscribes to the `EventApproved` logs emitted by `EventAdmission`, but it now must also listen for the two threshold-publication events (`NormThresholdsPublished` and `CosineThresholdsPublished`) at the start of each epoch. Upon seeing those events,  $S_j^1$  caches the new values of  $L_t$ ,  $B_t$ , and  $\tau_{\text{max}}$  in its local state, making them available to all subsequent verification calls in epoch  $t$ .

When a candidate bundle  $Z_{j'}^{t,\text{ext}} = (\Pi, \Sigma, \Gamma, \dots)$  arrives,  $S_j^1$  now executes its validation logic in three successive checks, all within the same transaction and using only on-chain precompiles:

1. A Groth16 verification of  $\Pi$  against the committed model and test commitments  $C_{j'}^{t,\text{model}}$  and

$C_{j'}^{t,\text{test}}$ , ensuring correct inference and loss computation.

2. A Bulletproof verification of  $\Sigma$ , using  $C_{j'}^{t,\text{model}}$  to guarantee that the weight update's  $\ell_2$ -norm lies in  $[L_t, B_t]$ .
3. A SNARK verification of  $\Gamma$ , using the public probe commitment  $C_{\text{probe}}$  to enforce  $\cos(z_{\text{old}}, z_{\text{new}}) \leq \tau_{\text{max}}$ .

If—and only if—all three checks succeed,  $S_j^1$  emits the usual `ProofOK` event and marks the block as valid; any failure causes an immediate reject. Because thresholds are advanced by the oracle-signed events and stored in contract state, no additional transactions or on-chain parameters are required.

After collecting the `ProofOK` events for all tips  $D_{j'}^T(t)$  received in epoch  $t$ ,  $S_j^1$  retrieves their associated loss values  $\mathcal{L}_{j'}^t$ , ranks the valid blocks in ascending order of  $\mathcal{L}$ , and selects the top  $K_V$  lowest-loss blocks to form  $\mathcal{D}_j^V(t)$ . The parameter  $K_V$  is a protocol-level constant that determines how many parent contributions each node adopts. Once  $\mathcal{D}_j^V(t)$  is finalized, node  $j$  proceeds to the submission stage with those selected parents.

All other EDSCs ( $S_j^2$  through  $S_j^5$ ) and the `EventAdmission/Oracle` workflow remain unchanged.

## 5.4 The ZK-HybridFL Procedure and Analysis

### 5.4.1 Workflow of ZK-HybridFL

This subsection outlines the workflow of ZK-HybridFL from the viewpoint of node  $j$  at epoch  $t$ . It describes how node  $j$  performs local training, commits its state, generates proofs, submits blocks, and participates in the on-chain verification, challenge, and aggregation stages.<sup>3</sup> These four stages repeat until the model converges.

---

<sup>3</sup>All on-chain triggers in Stages 2–4 result from threshold-signed `EventApproved` logs emitted by the `EventAdmission` contract. These logs are produced only after off-chain validation by the Oracle Committee (see Sec. 5.3.2). References to “approved events” in this workflow refer to those logs.

## Stage 1: Training and Proof Certification

- a. **Local training.** Node  $j$  downloads the latest global model  $\tilde{\mathbf{W}}_j^{t-1}$  from the side-chain and runs  $R$  SGD iterations on its private training batch  $\mathcal{D}_j^{t,\text{train}}$  (mini-batch size  $B$ ), yielding updated weights  $\mathbf{W}_j^t$ .
- b. **Commit.** It posts two KZG commitments

$$C_j^{t,\text{model}} = \text{Commit}(\mathbf{W}_j^t), \quad C_j^{t,\text{test}} = \text{Commit}(D_j^{t,\text{test}})$$

to the side-chain, together with a Lamport timestamp.

- c. **Proof generation (off-chain).** Using the universal proving key  $\text{pk}$ , node  $j$  evaluates the inference circuit on its private witness  $(\mathbf{W}_j^t, D_j^{t,\text{test}})$  and produces the non-interactive Groth16 proof  $\Pi_t^j$ . It then assembles the proof bundle  $Z_t^j = (\Pi_t^j, \mathcal{Y}_j^t, \mathcal{L}_j^t, C_j^{t,\text{model}}, C_j^{t,\text{test}})$  and buffers it locally until submission.
- d. **Block construction.** Once  $\Pi_t^j$  is ready, node  $j$  forms the block  $D_j(t) = [\mathbf{W}_t^j, Z_t^j]$  for insertion into the DAG.

## Stage 2: Block Submission

- a. **Fetch tips.** Node  $j$  reads the latest approved events to reconstruct the current tip set  $\mathcal{D}_j^T(t)$ , and pulls each tip's commitments  $C_{j'}^{t',\text{model}}, C_{j'}^{t',\text{test}}$  and the global verification key  $\text{vk}$  from the side-chain.
- b. **Verification.** Node  $j$  invokes the Validation Contract  $S_j^1$ , which for each candidate block performs

$$\text{Verify}(\text{vk}, \Pi_{j'}^{t'}, C_{j'}^{t',\text{model}}, C_{j'}^{t',\text{test}}, \mathcal{Y}_{j'}^{t'}, \mathcal{L}_{j'}^{t'})$$

and returns the top- $K_V$  verified parents  $\mathcal{D}_j^V(t)$ .

- c. **Publish block.** With  $\mathcal{D}_j^V(t)$  confirmed, node  $j$  calls  $S_j^2$ , which (i) records the chosen parents, (ii) embeds  $D_j(t)$  into the DAG, and (iii) marks it as a new tip.

### Stage 3: Consensus-Driven Confirmation & Challenge

- a. Node  $j$  syncs with the sidechain, incorporating new blocks  $D_{j'}(t)$  whose approved events have appeared on-chain.
- b. Upon synchronization,  $S_j^2$  advances to its second phase: it updates block statuses and records which blocks transition from unconfirmed to confirmed in epoch  $t$ .
- c. Node  $j$  executes the GRA on the DAG.
- d. Based on the GRA output, node  $j$  may trigger the Challenge Contract  $S_j^3$  to dispute the validity of certain blocks.
- e. If triggered,  $S_j^3$  stakes tokens from node  $j$  and notifies the oracles to adjudicate the dispute.
- f. Depending on the oracles' consensus (cf. Sec. 5.2.3), either the DAG is updated or node  $j$ 's stake is slashed.

### Stage 4: Global Model Aggregation

- a. Node  $j$  calls the Aggregation Contract  $S_j^4$  with the list of confirmed blocks from epoch  $t$ .
- b.  $S_j^4$  retrieves the corresponding models from the DAG and computes the new global model  $\tilde{\mathbf{W}}_j^t$ , storing it on the side-chain.
- c. Finally, node  $j$  invokes the Reward Contract  $S_j^5$ , which disburses tokens to node  $j$  based on its contributed loss  $\mathcal{L}_j^t$  (cf. Sec. 5.3.1).

At this point, node  $j$  holds the updated global model  $\tilde{\mathbf{W}}_j^t$  and begins the next epoch's local training, returning to Stage 1.

*Remark 5:* The four-stage protocol described in Sec. 5.4.1 remains intact; we simply augment Stages 1 and 2 to consume the committee-published scalars  $[L_t, B_t]$  and  $\tau_{\max}$ . At the boundary between epochs  $t - 1$  and  $t$ , each node  $j$  retrieves the signed events `NormThresholdsPublished` and `CosineThresholdsPublished` (Sec. 5.3.3) and caches the new bounds before beginning Stage 1. After local training and KZG commitments, Stage 1 proof generation now emits three non-interactive proofs  $\Pi_j^t$  (Groth16 for correct inference and loss),  $\Sigma_j^t$  (Bulletproof enforcing  $L_t \leq \|\Delta W_j^t\|_2 \leq B_t$ ), and  $\Gamma_j^t$  (Groth16 sub-proof enforcing  $\cos(z_{\text{old}}, z_{\text{new}}) \leq \tau_{\max}$  on the fixed probe set), which are concatenated into the extended bundle  $Z_j^{t,\text{ext}}$ . Stage 2's Validation Contract  $S_j^1$  then runs the three-step verification (Groth16  $\Pi$ , Bulletproof  $\Sigma$ , cosine  $\Gamma$ ) as in Sec. 5.3.3 and returns the top- $K_V$  parents exactly as before. Stages 3 and 4 continue to react only to the usual `ProofOK` events and require no modifications.

*Remark 6:* In ZK-HybridFL, the network can be effectively divided into two types of nodes—full nodes and light nodes based on their resource capabilities. This division is motivated by the need to accommodate diverse participant environments, optimize system performance, and ensure scalability. Full nodes, with ample computational power and storage, maintain the complete global DAG ledger, comprehensive sidechains, and carry out resource-intensive tasks, including oracle functions. These nodes handle heavy operations such as ledger maintenance, smart contract execution, and dynamic event validation, which are crucial for the integrity and reliability of the overall system.

In contrast, light nodes operate with lighter versions of the DAG and sidechain, focusing on essential local training and basic participation in the FL process. By delegating complex, resource-heavy tasks to full nodes, light nodes can efficiently contribute to model updates and generate proofs without the burden of maintaining an entire global state. This stratification not only leverages the strengths of more capable participants but also enables a wide range of devices with limited resources to join the network. The oracle functionality, consistent across both node types, serves as a trusted intermediary, validating events and bridging interactions between light and full nodes. Fig. 5.2 illustrates the architectural division between full nodes and light nodes in ZK-HybridFL. Also, the overall interaction between the trainers, the oracle committee, the sidechain

smart contracts, and the DAG ledger is illustrated in Fig. 5.3, including both the main training and challenge phases.

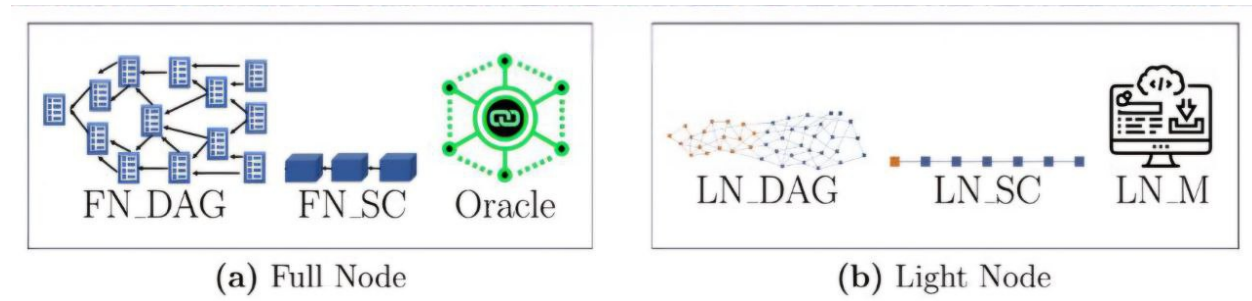


Figure 5.2: Full nodes (FNs) maintain a complete DAG (FN\_DAG) and sidechain (FN\_SC) while handling oracle functions. In contrast, light nodes (LNs) use a trimmed DAG (LN\_DAG) and lightweight sidechain (LN\_SC), with LN\_M serving as LN’s core controlling module.

#### 5.4.2 Comparison with ChainFL

Having outlined the workflow of ZK-HybridFL, we now direct our analysis toward ChainFL. While Blade-FL employs blockchain to decentralize FL, ChainFL enhances this approach by integrating a DAG with a sharded architecture. This refined design not only addresses the scalability challenges inherent to blockchain-based systems but also provides a more efficient decentralized framework, making ChainFL the appropriate benchmark for our evaluation. Both ZK-HybridFL and ChainFL share similar guiding objectives: they eliminate the reliance on a central server, adopt DAG-based structures to mitigate the linear block-generation bottlenecks of traditional blockchain ledgers. These broad convergence reflect a shared desire to accommodate large-scale FL among edge or IoT devices, and deliver better security in adversarial or untrusted settings.

In this section, we analyze both schemes from two perspectives; first, the *learning perspective*, focusing on model validation strategies, resilience against adversarial updates, and overall learning efficiency; and the *distributed ledger perspective*, focusing on consensus mechanisms, block selection strategies, and scalability. To ground our comparison, we first define two problematic node behaviors: *adversarial* nodes and *lazy* nodes. Adversarial nodes inject subtle noise or degraded parameters into their local model updates, ensuring they can pass superficial validation, i.e., in case of using a public

dataset the model performs well on the public dataset, but gradually corrupt the global model.

Our threat model follows the “utility-preserving” model-poisoning adversary of [207]–[209]. Concretely, let  $(\mathbf{W}, D_{pub})$  denote the classification accuracy of model  $\mathbf{W}$  on the public validation set  $D_{pub}$ . We call node  $i$  adversarial at round  $t$  if its local update  $\mathbf{W}_i^t$  simultaneously (i) attains nearly the same public-set accuracy as an honest update, i.e.  $(\mathbf{W}_i^t, D_{pub}) \geq (\mathbf{W}_{\text{honest}}^t, D_{pub}) - \varepsilon$  for a small slack  $\varepsilon$  (we use  $\varepsilon = 0.01$ ), and (ii) is chosen to pull the federated average  $\tilde{\mathbf{W}}^t = \sum_j \omega_j \mathbf{W}_j^t$  as far as possible (in  $\ell_2$ -norm) from the honest-only average  $\tilde{\mathbf{W}}_{\text{honest}}^t = \sum_j \omega_j \mathbf{W}_{j,\text{honest}}^t$ , i.e. to maximize the “drift”  $\|\tilde{\mathbf{W}}^t - \tilde{\mathbf{W}}_{\text{honest}}^t\|_2$ . Such updates “look good” on  $D_{pub}$  but still cumulatively degrade the global model. a behavior quantified in Sec. 5.5.4.

Lazy nodes reduce computational effort by skipping training in some epochs and resubmitting previous updates, slowing convergence and polluting aggregation. Inadequate validation allows these behaviors to persist undetected, undermining the integrity and efficiency of the entire FL process.

The threat analysis in this section is intentionally framed for the *baseline ZK-HybridFL pipeline* described, that is, the version employing the core Groth16 proof bundle without the additional norm-range Bulletproof and embedding-cosine SNARK. This is the variant we compare against Blade-FL and ChainFL in the main body of the chapter, since those schemes offer no counterpart to the extended checks. The *extended-ZKP variant*, which augments every block with two additional proofs, is analyzed in detail in Supplementary C, including attacks it prevents and formal proofs of its security benefits.

## Learning Perspective

From the learning standpoint, a fundamental difference arises in how ChainFL and ZK-HybridFL validate model updates and aggregate them into a global model. ChainFL partitions nodes into shards, each governed by a SLN that uses Raft consensus to synchronize local training. Once the shard’s local model is aggregated, the SLN periodically attaches this model to a mainchain. When a shard needs to integrate models from others, it selects updates from the DAG’s *tip* set on the

mainchain, relying on a public reference dataset to assess accuracy or loss. While this approach eliminates the single global aggregator, it has major problems.

First, when a public dataset is used for validating models this approach creates vulnerabilities when lazy nodes are present. A lazy node that resubmits unchanged parameters from a previous epoch may still meet the subchain’s performance threshold if its model had acceptable accuracy on the public dataset. Over time, this degrades the effectiveness of the FL process by slowing convergence and increasing redundancy in the model aggregation step.

Second, due to the fact that ChainFL uses the tip models for aggregation as introduced in Sec. 5.2.3, a malicious node can exploit this by performing an orphanage attack. Using the tip models for global aggregation means nodes are obtaining the global models that network has not reached consensus on its validity. Thus, in an orphanage attack, the attacker node deliberately chooses some of their own previously submitted blocks as parents for new blocks, affecting how the new global model is constructed, as this new global model becomes an amalgamation of both valid models and compromised models. Then, the local updated model, trained on this amalgamated global model, may still pass validation checks using a public dataset; however, its degraded quality will gradually undermine the overall integrity and performance of the network. Over time, this cycle prevents the network from converging toward a high-quality model and may cause it to settle at a low-performance, locally optimal state from which it cannot recover (see Sec. V-C-1).

By contrast, ZK-HybridFL removes any reliance on a public reference dataset by enforcing an inference–validation pipeline built on KZG commitments and non-interactive SNARKs. Rather than aggregating every tip that happens to be visible, the protocol only ever considers confirmed blocks whose proofs have passed on-chain verification. In each epoch  $t$ , trainer  $j$  first publishes the KZG commitments  $C_j^{t,\text{model}} = \text{Commit}(\mathbf{W}_j^t)$  and  $C_j^{t,\text{test}} = \text{Commit}(D_j^{t,\text{test}})$ , irrevocably binding its flattened weight vector  $\mathbf{W}_j^t$  and private test batch  $D_j^{t,\text{test}}$ . It then generates a Groth16 proof  $\Pi_j^t$  attesting that the committed model achieves the stated loss on the committed data. Only after that proof is verified on-chain and the block’s aggregated weight exceeds the network threshold does the update influence the global model.

**Lazy-node defence.** If a trainer attempts to re-submit its previous weights  $\mathbf{W}_j^t$  in epoch  $t + 1$ , the new commitment  $C_j^{t+1,\text{model}}$  will be bit-for-bit identical to  $C_j^{t,\text{model}}$ . The validation contract  $S_j^1$  detects the duplicate and discards the block before it can be ranked, so stale updates never propagate.

**Adversarial accuracy spoofing.** Two natural attack vectors are neutralized. First, if an attacker proves with a degraded model  $\widetilde{\mathbf{W}}$ , the resulting high loss is visible in the proof bundle and the block is filtered out during parent selection. Second, if an attacker runs inference with honest weights  $\mathbf{W}$  but publishes a commitment  $C_j^{t,\text{model}} = \text{Commit}(\widetilde{\mathbf{W}})$ , the KZG opening contained in the proof cannot match that commitment. On-chain verification therefore fails and the block is rejected. Because invalidated blocks carry no aggregated weight, only fresh, correctly computed updates ever enter the global aggregation. This enforcement provides significantly stronger robustness than ChainFL, as we empirically demonstrate in Sec. V-C-1.

## Distributed Ledger Perspective

The ledger architecture and consensus mechanism are critical to system performance in terms of latency, scalability, and throughput. In ChainFL, the use of Raft-based subchains centralizes coordination in SLNs, which manage local shard operations and require explicit cross-shard synchronization. This centralization introduces several drawbacks. If an SLN becomes adversarial or is overwhelmed, its corresponding shard suffers from degraded performance, which in turn creates a bottleneck for the entire network. Such centralization not only poses security risks—since a malicious SLN can manipulate or delay the aggregation and propagation of updates—but also increases latency because the coordinated, synchronous agreement required by Raft consensus limits throughput and scalability.

In contrast, ZK-HybridFL is designed from the ground up to combine DAG-based concurrency with cryptographic verification, sidechain-based event logic, and oracle-assisted validation. This holistic co-design offers several advantages over ChainFL. First, by offloading resource-intensive tasks such as ZKP verification to dedicated sidechains governed by event-driven smart contracts, ZK-HybridFL executes its consensus effectively. Second, the decentralized validation distributed

Aspect	ChainFL	ZK-HybridFL
<b>Learning Perspective</b>		
<i>Validation &amp; Aggregation</i>	<ul style="list-style-type: none"> <li>• Public dataset</li> <li>• Tip-based selection</li> </ul>	<ul style="list-style-type: none"> <li>• ZKPs</li> <li>• Aggregation from confirmed models</li> </ul>
<i>Handling Lazy Nodes</i>	<ul style="list-style-type: none"> <li>• Unchanged model can pass if accuracy remains acceptable</li> </ul>	<ul style="list-style-type: none"> <li>• Committed KZG checks prevent replay</li> </ul>
<i>Handling Adversarial Updates</i>	<ul style="list-style-type: none"> <li>• Subtle noise can slip through</li> <li>• Orphanage attacks exploit tip models</li> </ul>	<ul style="list-style-type: none"> <li>• Inference-model mismatch triggers rejection</li> <li>• Insufficient AW halts malicious updates</li> </ul>
<b>Distributed Ledger Perspective</b>		
<i>Consensus Design</i>	<ul style="list-style-type: none"> <li>• Raft-based subchains with leader nodes</li> <li>• Requires cross-shard sync</li> </ul>	<ul style="list-style-type: none"> <li>• DAG with sidechain event logic</li> <li>• Oracles and EDSCs</li> </ul>
<i>Scalability &amp; Performance</i>	<ul style="list-style-type: none"> <li>• SLN is a bottleneck</li> <li>• Centralized coordination increases latency</li> </ul>	<ul style="list-style-type: none"> <li>• Decentralized validation</li> <li>• Lower latency, higher throughput</li> </ul>

Table 5.3: Comparison of ChainFL and ZK-HybridFL.

among multiple oracles and smart contracts avoids the large-scale coordination lags inherent to a multi-layer, leader-based mechanism, thereby keeping the DAG agile and uncluttered. Then this architecture not only reduces latency and improves throughput but also scales more efficiently with network size. Each contract is triggered only under specific circumstances, diminishing reliance on any single trusted node. Overall, ZK-HybridFL’s distributed ledger design delivers a more granular security model and higher performance, establishing clear advantages over the SLN-dependent, Raft-based approach of ChainFL. We will demonstrate the superior performance of ZK-HybridFL over ChainFL in the context of distributed ledger through simulations provided in the supplementary material.

Table 5.3 summarizes the analysis between ZK-HybridFL and ChainFL from learning and distributed ledger perspectives.

## 5.5 Simulation Results

In this section we first describe the simulation setup and then present our experimental results, comparing our proposed ZK-HybridFL approach with Blade-FL and ChainFL.

### 5.5.1 Simulation Setup

To simulate Blade-FL and ChainFL, we used their implementations from the respective GitHub repositories [210] and [211], making necessary modifications to suit our simulation requirements. For our proposed scheme the FL process is built using TensorFlow Federated (TFF) [212] to simulate local model training across distributed nodes. Training with a mini-batch of size  $B$  consist of  $R$  GSD iterations per FL epoch. This setup simulates varying computational loads on each node.

For decentralized coordination, the blockchain architecture integrates a DAG ledger, simulated using the GoShimmer [112] framework, to manage parallelized model update submissions. Additionally, a Substrate-based [163] sidechain is employed to handle the core smart contracts responsible for verifying ZKPs, aggregating models, resolving challenges, and distributing rewards. The use of WebAssembly (Wasm)-based [213] smart contracts ensures efficient and low-overhead execution of decentralized logic, including ZKP verification and global model aggregation. The integration between the DAG and the Substrate blockchain is achieved via gRPC [214], facilitating fast and secure communication, with Apache Kafka [215] serving as a message broker to manage event-driven architecture. ZKPs are generated using EZKL [216], and zkML [217] proof systems.

We denote  $n$  as the total number of nodes,  $\gamma$  as the percentage of lazy nodes, and  $\mu$  as the percentage of adversarial nodes. The numbers of lazy and adversarial nodes are then  $\lceil n\gamma \rceil$  and  $\lceil n\mu \rceil$ , respectively, where  $\lceil \cdot \rceil$  is the rounding operation. Lazy nodes in all three schemes are participants that avoid fresh local training and instead resubmit a model update from a previous epoch or multiple past epochs. On the other hand, adversarial nodes are modeled as follows. In Blade-FL, adversarial nodes collectively control 51% of the PoW computation, thereby dominating block creation and ensuring that their noisy updates are predominantly used in the next global model aggregation. It is important to note that this 51% refers to the total hashing power controlled by the adversarial nodes. For example, if  $\mu$  is set to 20%, then although only 20% of the nodes behave adversarially, they command 51% of the network's total PoW hashing power, allowing them to exert influence on the system. It is important to note that a smaller  $\mu$  means fewer but more powerful adversarial nodes, while a larger  $\mu$  distributes the 51% PoW among more adversarial nodes, increasing the diversity

and number of attacks manifested in noisy models. A lower  $\mu$  allows honest nodes to retain some influence, whereas a higher  $\mu$  accelerates model degradation.

In contrast, in both ChainFL and ZK-HybridFL, adversarial nodes implement a dual attack strategy: (i) they add noise to their locally trained update and (ii) they perform an orphanage attack by deliberately selecting their own (or colluding) previously submitted blocks as parent blocks when appending a new block to the DAG ledger. This results in a higher aggregation weight for adversarial blocks, thereby amplifying their influence on the formation of the subsequent global model.

Other system parameters are as follows: mini-batch size  $B = 50$ , number of local GSD iterations  $R = 5$  with learning rate  $\eta = 0.01$ , and number of the selected parents  $K_v = 4$  for each submitted block in ZK-HybridFL.

Moreover, to emulate real-world network conditions, the simulation includes nodes with varying bandwidths (ranging from 10 Mbps to 50 Mbps) and latencies (ranging from 50 ms to 200 ms). These parameters are intrinsic inputs required by the simulator. We set these values to realistic ranges to reflect heterogeneity in actual networks.

## 5.5.2 Learning Tasks

### Task 1: Image Classification

Task 1 focuses on classifying handwritten digits from the MNIST dataset, which comprises 70,000 grayscale images of size  $28 \times 28$ , distributed across ten classes (digits 0-9). A lightweight convolutional neural network (CNN) based on MobileNetV2 [218] is employed for this task. The evaluation metric is classification accuracy, i.e., the proportion of correct classifications over all test samples.

For Task 1, we reserve 10,000 images as a public dataset used solely by Blade-FL and ChainFL to validate updates. The remaining 60,000 images are evenly distributed among the  $n$  nodes, and each node partitions its assigned images into 80% for local training and 20% for local testing. By contrast, in ZK-HybridFL, there is no public dataset for validation; instead, all 70,000 images are evenly split among the  $n$  nodes, each node again applying an 80%-to-20% split between local

training and local testing, and designating 20% of its local test set as the private inference batch for generating zero-knowledge proofs.

Throughout the experiments, when we report testing accuracy, we refer to performance evaluated on each scheme’s local test sets (aggregated network-wide). In Blade-FL and ChainFL, the public dataset does not serve for final accuracy metrics but purely for validating updates; similarly in ZK-HybridFL, nodes rely on their private inference batches for zero-knowledge proof generation only rather than for final accuracy evaluation.

## Task 2: Text Sentiment Analysis

Task 2 involves predicting the next word in a sequence using a gated recurrent unit (GRU)-based model trained on the Penn Treebank dataset that comprises 345,526 tokens. The dataset contains diverse text sequences that capture linguistic dependencies, making it ideal for language modeling tasks. The output is a probability distribution over the vocabulary for the next word.

The model’s performance is evaluated using perplexity, which measures its ability to predict the correct next word. Perplexity is computed as  $e^{-\frac{1}{N} \sum_{i=1}^N \log(p_{y_i})}$ , where  $p_{y_i}$  is the predicted probability of the true next word  $y_i$  at position  $i$ , and  $N$  is the total number of predictions. Lower perplexity indicates better predictions, as the model assigns higher probabilities to the correct next words.

For Task 2, we similarly set aside 45,526 tokens as the public dataset for Blade-FL and ChainFL, distributing the remaining 300,000 tokens evenly among the  $n$  nodes. Each node uses 80% of its tokens for local training and 20% for local testing, and in ZK-HybridFL, 20% of that local test set is marked as a private inference batch.

### 5.5.3 Results and Analysis

#### Learning Perspectives

##### FL training convergence

Fig. 5.4 presents the training loss curves for three schemes: Blade-FL, ChainFL, and our proposed ZK-HybridFL, in a network consisting of  $n = 15$  nodes. In this simulation, 20% of the nodes behave adversarially, while 10% are lazy. Blade-FL demonstrates moderate performance during the initial epochs; however, it ultimately fails to converge. This shortcoming is primarily due to the adversaries, which eventually control approximately 51% of its PoW consensus. With this majority, adversarial nodes can dictate the block creation process, thereby contaminating the global model aggregated at each epoch. Consequently, the model quality degrades over time, and convergence to a low-loss, high-performance model is hindered.

ChainFL exhibits improved convergence compared to Blade-FL owing to its DAG-based architecture and sharded design. Nonetheless, its performance remains inferior to that of ZK-HybridFL. As detailed in Sec. 5.4.2, ChainFL's reliance on a public reference dataset for validation leaves it vulnerable to lazy nodes that merely resubmit previous updates without proper retraining. Additionally, adversarial nodes in ChainFL can exploit the tip-based model aggregation by manipulating the selection of blocks, thereby injecting subtle corruptions into the global model resulting in system being unable to converge to lower loss values.

In contrast, ZK-HybridFL benefits from an inference-validation pipeline that leveraging ZKPs. These mechanisms ensure that only genuine and updated models contribute to the global aggregation process. Even in the presence of lazy nodes and adversaries, our scheme systematically filters out invalid or stale updates. This is reflected in the steady reduction of the training loss over successive epochs, indicating continuous model improvement and robust convergence behavior.

## Invalid model detection

To gain further insights into the robustness of each scheme, Fig. 5.5 plots the number of detected invalid models corresponding to the results shown in Figs. 5.4. We define a model as detected invalid if it fails to meet the selection criteria for inclusion in the global aggregation process. Specifically, in Blade-FL, an update is invalid if, after being broadcast with its digital signature, it is not ultimately included in any newly mined block. In ChainFL, an update is invalid if it is not selected as a parent block on the DAG within a predetermined staleness time since it will be ineligible for parent selection in all subsequent epochs. Lastly, in ZK-HybridFL a model is invalid if it is revoked through the challenge mechanism.

Fig. 5.5 includes a reference horizontal line representing an ideal system that detects all invalid models. As expected, Blade-FL shows the poorest performance in invalid model detection, with a high number of undetected or late-detected invalid updates. ChainFL performs better than Blade-FL; however, its detection capability is still significantly lower compared to ZK-HybridFL.

Notably, our ZK-HybridFL scheme initially detects fewer invalid models than the other two approaches. This is because our invalidation process requires the network to reach consensus through a challenge mechanism before a model can be revoked. Consequently, while an invalid model might be added in one epoch, it may only be formally invalidated several epochs later. Despite this delayed detection, as training progresses, ZK-HybridFL effectively “catches up” in its detection performance. The gradual but consistent removal of invalid models contributes directly to the superior convergence behavior observed in Fig. 5.4.

The results in Fig. 5.4 and Fig. 5.5 corroborate our theoretical analysis in Sec. 5.4.2: by ensuring that only valid, high-quality model updates influence the global model, ZK-HybridFL not only accelerates convergence but also maintains model integrity in adversarial and lazy environments.

## Model performance vs number of nodes

Fig. 5.6 presents the testing accuracy versus the number of nodes for the three schemes with the adversarial and lazy ratios fixed at  $\mu = 15\%$  and  $\gamma = 15\%$ , respectively. For Task 1, the

proposed ZK–HybridFL scheme demonstrates a striking improvement as the number of nodes increases. Starting at an accuracy of 0.80 with 5 nodes, its performance climbs steeply to 0.90 at 10 nodes, 0.95 at 15 nodes, and nearly saturates at 0.98–0.99 for networks of 20 nodes and more. This behavior confirms that the robust inference–validation pipeline anchored by ZKP enables the scheme to effectively harness the increasing pool of valid updates from a larger network, thereby substantially mitigating the impact of both adversarial and lazy nodes. Importantly, the success of the ZKP mechanism is not achieved in isolation; it is realized through the architectural choices of ZK–HybridFL. The concurrent DAG structure facilitates decentralized and efficient propagation of updates, while the sidechain-based event-driven logic offloads the computationally intensive tasks of generating and verifying proofs. Together, these elements create a synergistic framework in which the ZKP-based validation can operate effectively, ensuring that only high-quality, genuinely updated models are aggregated.

In contrast, ChainFL’s accuracy increases gradually from 0.55 with 5 nodes to 0.70 with 30 nodes. Although the DAG-based, sharded architecture helps dilute the effect of malicious contributions, ChainFL’s reliance on a public validation dataset still leaves it somewhat susceptible to lazy behavior. Consequently, its performance gains with more nodes are more subdued compared to the proposed scheme.

Blade–FL, on the other hand, suffers significantly as the network grows. Its accuracy is low from the outset—starting at 0.45 with 5 nodes—and further deteriorates to 0.33 at 30 nodes. This decline highlights the vulnerability of the PoW–based consensus mechanism in Blade–FL, where even a fixed fraction of adversaries can increasingly compromise the global model as more nodes participate.

For Task 2, the proposed ZK–HybridFL scheme achieves the lowest perplexity, i.e., the best performance starting at 145.73 with 5 nodes and steadily decreasing to 117.67 with 30 nodes. This improvement reflects its superior ability to aggregate high-quality updates and filter out unreliable contributions, ensuring better language model convergence.

In contrast, ChainFL exhibits a more moderate decline in perplexity, starting at 218.59 with

5 nodes and improving to 167.08 at 30 nodes. While the architecture helps alleviate the effects of adversarial updates to some extent, its reliance on a public validation dataset leaves it more susceptible to compromised updates.

Meanwhile, Blade-FL performs the worst, showing an increasing perplexity as more nodes are added. Beginning at 284.17 with 5 nodes, its perplexity worsens to 406.89 at 30 nodes. This upward trend highlights the detrimental effects of the PoW-based consensus mechanism, where even a fixed proportion of adversaries can degrade model quality more significantly as the network expands.

Thus, Fig. 5.6 reinforces the earlier analysis: while ChainFL can partially mitigate adverse behavior, only the proposed ZK-HybridFL scheme fully leverages an increasing number of nodes to enhance model performance. By integrating a flexible DAG structure and sidechain-assisted validation, it ensures that only high-quality updates contribute to model aggregation.

### **Model performance vs percentage of adversarial nodes**

Fig. 5.7 illustrates the testing accuracy for Task 1 and perplexity for Task 2 of the three schemes as the percentage of adversarial nodes increases from 5% to 30%, with the number of nodes  $n = 15$  and the lazy node ratio  $\gamma = 15\%$ .

For Task 1, ChainFL's performance degrades gradually from an accuracy of 0.70 at 5% adversaries down to 0.50 at 30%. This steady decline reflects its vulnerability stemming from the reliance on a public reference dataset and tip-based update selection, which makes it susceptible to both lazy nodes resubmitting outdated models and adversarial nodes injecting subtle degradations. Blade-FL, on the other hand, starts with a lower baseline of 0.50 at 5% adversaries and drops significantly to 0.30 at 30%. This marked deterioration is indicative of the inherent weakness of its PoW-based consensus mechanism, which adversaries can exploit more easily as their proportion increases.

In contrast, the proposed ZK-HybridFL scheme exhibits remarkable resilience, achieving near perfect accuracy of 0.99 at low adversary levels of 5% and a slight reduction to 0.88 at 30%. This robustness can be attributed to its inference-validation pipeline based on ZKP, which ensures that

only genuinely updated models, confirmed through cryptographic verification and consensus on their AW, are used in the global model. Such a mechanism effectively counters both lazy behavior and adversarial attacks, preventing the subtle propagation of corrupted updates.

For Task 2, we observe a perplexity-based trend that mirrors the accuracy degradation seen in Task 1. The proposed ZK-HybridFL scheme maintains the lowest perplexity, starting at 117.18 at 5% adversaries and only increasing slightly to 120.92 at 30%, demonstrating its ability to consistently aggregate high-quality model updates despite adversarial interference.

ChainFL, however, shows a worsening perplexity, increasing from 174.87 at 5% adversaries to 218.59 at 30%. This increase highlights ChainFL's limitations in filtering out degraded updates, as adversaries introduce inconsistencies into the language model.

Meanwhile, Blade-FL exhibits the most severe performance degradation, with perplexity worsening from 270.64 at 5% adversaries to 378.89 at 30%. This significant rise underscores the fundamental weakness of Blade-FL's PoW-based consensus, which fails to prevent adversarial nodes from injecting disruptive updates, leading to degraded model convergence.

These results underscore the critical advantage of ZK-HybridFL's dual-layer validation approach over the strategies employed by ChainFL and Blade-FL. By forgoing a public dataset for validation and avoiding reliance on tip-based model aggregation, ZK-HybridFL minimizes the risk of orphanage attacks and the inadvertent inclusion of stale or malicious updates.

### **Model performance vs percentage of lazy nodes**

Fig. 5.8 presents the testing accuracy for Task 1 and perplexity for Task 2 as the ratio of lazy nodes increases from 5% to 30%, with  $n = 15$  nodes and a fixed adversary ratio of  $\mu = 15\%$ .

Lazy nodes that simply resubmit previous model updates without retraining can significantly hinder convergence by contaminating the aggregation process with stale information. For Task 1, ChainFL's accuracy drops from 0.73 to 0.60 and Blade-FL's from 0.55 to 0.42 as lazy node proportions increase. In contrast, ZK-HybridFL filters out stale updates, maintaining high accuracy that only slightly decreases from 0.99 to 0.94.

For Task 2, since ChainFL validates updates using a public reference dataset, stale submissions by lazy nodes may still pass the performance threshold, leading to a moderate increase in perplexity from 169.65 at 5% lazy nodes to 199.42 at 30%. Similarly, Blade-FL, already burdened by its susceptibility to adversarial influence, suffers an even sharper degradation, as perplexity rises from 241.85 to 307.21 as the proportion of lazy nodes increases.

In contrast, the proposed ZK-HybridFL scheme demonstrates resilience against lazy nodes. Its committed hash comparisons to detect and dismiss duplicate or outdated model submissions help identify stale models. This sidechain mechanism ensures that only freshly updated, valid contributions are aggregated into the global model. As a result, ZK-HybridFL maintains a low perplexity, increasing only slightly from 117.18 to 123.55 as the lazy node ratio rises.

These results highlight the effectiveness of ZK-HybridFL’s sidechain-based validation mechanism, which prevents stale updates from polluting the aggregation process, ensuring that only genuinely updated models contribute to training.

Additional simulations and discussions on the distributed ledger perspectives, specifically analyzing the latency, throughput, and scalability of ZK-HybridFL, are provided in the supplementary material accompanying this chapter.

### **Zero-Knowledge Proof Cost and Scalability**

A central concern for ZK-HybridFL is the overhead of generating SNARKs for each node’s private validation batch. While on-chain SNARK verification is efficient, proof generation, especially for modern neural networks, can become a practical bottleneck. To provide a detailed, quantitative picture, we benchmarked both proof and verification on the exact models used in our experiments, using a 16-core Intel Xeon Gold 6338 (3.0 GHz, 64 GB RAM) for CPU runs and an NVIDIA A100 80 GB for GPU-accelerated proofs. In all cases, each node constructs a proof over a local test mini-batch of size  $B = 10$  samples, balancing statistical rigor against proof latency.

Table 5.4 summarizes the proof-generation time, GPU speedup, and proving-key memory footprint for each model used in our benchmarks.

Table 5.4: Proof-generation cost per node (batch size  $B = 10$ ).

Model	Params ( $\times 10^3$ )	FLOPs/sample ( $\times 10^3$ )	CPU prove (s)	GPU prove (s)	PK size (GB)
MLP-3k	3.6	3.5	3.2	0.51	0.16
CNN-20k	19.8	68	32	4.1	4.2
MobileNetV2-0.5	1,300	$3 \times 10^5$	665	76	31
GRU-256	29	950	360	44	4.3

Proof-generation time and memory footprint both scale roughly linearly with the per-inference FLOPs of the circuit. A small MLP with  $\approx 3.6 \times 10^3$  parameters and  $3.5 \times 10^3$  FLOPs per sample completes proof generation in only 3.2 s on CPU and 0.51 s on GPU, requiring just 0.16 GB of RAM for the proof key. In contrast, a GRU-256 (approximately  $2.9 \times 10^4$  parameters and  $9.5 \times 10^5$  FLOPs) takes 360 s on CPU or 44 s on GPU, with a 4.3 GB proof key. The largest circuit, MobileNetV2-0.5 (approximately  $1.3 \times 10^6$  parameters and  $3 \times 10^8$  FLOPs), demands 665 s on CPU or 76 s on GPU, and a 31 GB proof key. We further observed linear scaling of GPU proof time with batch size: for the GRU-256,  $t_{\text{prove}} \approx 4.4 B \pm 0.35$  s on GPU for  $B$  up to 40.

SNARK verification, by contrast, incurs only trivial overheads. Verification on CPU completes in under 0.11 s for all models; serialized proof sizes range from 17 to 118 KB, and on-chain gas costs lie between 19 and 24 thousand units on our Substrate test network. At a typical block gas limit of  $2 \times 10^6$  units, publishing proofs for all 30 participants consumes less than 1% of block capacity, and storage fees amount to only  $\sim 5 \times 10^{-5}$  token per proof, which is negligible relative to our reward mechanism.

Table 5.5: Proof verification overhead.

Model	Verify time (s)	Proof size (KB)	Gas (k units)
MLP-3k	0.021	21	19
CNN-20k	0.028	17	21
MobileNetV2-0.5	0.095	118	24
GRU-256	0.108	41	23

The two extra proofs introduced in Sec. 5.2.4 add only a modest cost on top of the base Groth16 check. Table 5.6 reports generation time, proof size, and on-chain gas for (i) the Bulletproof  $\Sigma_j^t$  that enforces  $L_t \leq \|\Delta W\|_2 \leq B_t$  and (ii) the embedding-cosine SNARK  $\Gamma_j^t$ . Because Bulletproof proof

Table 5.6: Incremental proofs: generation and verification cost (same hardware as Tables 5.4–5.5).

<b>Proof</b>	<b>CPU prove</b>	<b>GPU prove</b>	<b>Proof size</b>	<b>Verify gas</b>	<b>Dep. on <math>n</math></b>
$\ell_2$ -BP, MLP-3k	0.07 s	0.008 s	1.6 kB	38 k	linear
$\ell_2$ -BP, CNN-20k	0.4 s	0.05 s	2.3 kB	38 k	linear
$\ell_2$ -BP, GRU-256	0.6 s	0.08 s	2.8 kB	38 k	linear
$\ell_2$ -BP, MobileNetV2-0.5	26 s	2.6 s	8.1 kB	38 k	linear
Cosine-SNARK (all models)	2.4 s	0.30 s	0.2 kB	25 k	constant

time grows *linearly* with the number of model parameters  $n$  (and the proof itself only  $O(\log n)$ ), the worst-case MobileNetV2-0.5 update still completes in  $\approx 26$  s on CPU or 2.6 s on GPU. The cosine check is model-agnostic; its tiny circuit produces a 0.2 kB Groth16 proof in  $\approx 2.4$  s (CPU) or 0.30 s (GPU). Verification costs are fixed—38 k gas for  $\Sigma$  and 25 k gas for  $\Gamma$ —bringing the *total* per-update gas from  $\sim 61$  k to  $\sim 124$  k, i.e., still 5 % of a standard 2 M-gas block for 30 simultaneous trainers. Overall, the extended Z-bundle remains well within practical limits: proof generation overlaps with training (“predict-then-prove”), while on-chain verification still occupies 5 % of block gas and adds no measurable latency to consensus.

Models requiring more than 24 GB of GPU memory for their proof keys leverage `tensorplonk`’s 512 MiB paging to stream the key from CPU RAM (up to 64 GB) at a minor ( $< 5\%$ ) performance penalty. As an alternative, recursive folding (i.e., Halo Infinite) partitions large circuits into sub-proofs with sub-20 GB keys, recombining them on-chain with only  $\approx 20$  ms of additional verification time.

These results demonstrate that proof generation is the principal scaling challenge in ZK-HybridFL, yet even million-FLOP models can be proved in under two minutes on modern GPUs. SNARK verification and on-chain gas/storage remain negligible compared to consensus delays, and the “predict-then-prove” strategy fully overlaps SNARK computation with training, preserving ZK-HybridFL’s end-to-end epoch latencies (8–52 s as reported in Sec. V-A) without compromise.

#### 5.5.4 Additional Simulations

##### **Analysis of Stake-Weighted Aggregation**

To provide the quantitative evidence about benefits of the stake-weighted aggregation proposed in ZK-HybridFL, we perform an ablation study comparing our on-chain stake-weighted aggregation rule against a vanilla uniform average under the same node pool, data partitions, and fault rates described in Sec. 6.5.3. Fig. 5.9 plots the loss, accuracy, and perplexity curves for both aggregation rules on two canonical tasks, and reports fault tolerance under a mixed adversarial and lazy client setting.

In the federated vision experiment on MNIST (Task 1), stake weighting drives rapid convergence: the training loss drops from 1.41 to 0.12 within 100 epochs, whereas uniform averaging plateaus near 0.30. This translates to a final accuracy of 97% under stake weighting, compared to 91% with uniform averaging, which is a gap of 6.6 percentage points (Fig. 5.9a). Similarly, in the language modelling experiment on Penn Treebank (Task 2), stake-weighted aggregation achieves a test perplexity of around 118 at convergence, approximately 12 points lower than uniform averaging (Fig. 5.9c), indicating consistently stronger generalization.

To stress test robustness, we introduce a mixed-fault regime in which 30% of clients are faulty (evenly split between Byzantine adversarial and lazy behaviors,  $\gamma = \mu = 0.15$ ). Under this setting, uniform averaging suffers a sharp drop in test accuracy to around 72%, whereas stake weighting retains around 88% final accuracy (Fig. 5.9d). This improvement stems from our dynamic slashing mechanism: any client update failing its ZK proof is rejected and that client’s stake is halved ( $\lambda = 0.5$ ), while honest contributors accrue stake at rate  $\eta = 0.05$  per accepted update before normalization.

##### **Extended-ZKP Threat Model and Simulation Setup**

In Sections II–IV we introduced two lightweight ZKP checks: a Bulletproof enforcing that each client’s update norm stays within a dynamic window  $[L_t, B_t]$ , and an embedding-cosine SNARK

ensuring the model change is sufficiently large in feature space. These are designed to catch subtle “stealth” updates that pass the main Groth16 proof yet contribute almost no learning. In our original simulations (Sec. V-C) we already exercised the exact-replay lazy attack, which is detected by a hash collision at the Groth16 stage. Here we focus on three new variants that all pass Groth16 but should be flagged by the extended bundle. When comparing against ChainFL and BladeFL, we retain the same lazy and adversarial behaviors from Sec. V-A, effectively giving those protocols a less stringent attack profile, since they do not implement ZKP-based replay, norm, or cosine checks.

First, we model a *perturb-replay lazy* node. It resubmits a previous weight tensor  $\mathbf{W}_j^{t-\Delta}$  (with  $\Delta$  sampled uniformly from  $\{1, 2, 3\}$ ) plus a tiny noise vector  $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$ , where  $\sigma = 10^{-5} \|\mathbf{W}\|_2$ . This ensures the update hash differs while  $\|\varepsilon\|_2 \ll L_t$ , so it bypasses replay detection but should violate the Bulletproof lower-bound check.

Next, in the *minimal-norm stalling* attack, a node runs one honest SGD step  $\delta\mathbf{W}$  and then rescales it by  $\alpha = L_t / \|\delta\mathbf{W}\|_2$ , so it exactly meets the  $\ell_2$  bounds without meaningful progress.

Finally, the *semantic-stalling* attack performs full local training to produce  $\mathbf{W}_{\text{tmp}}$ , cherry-picks (or synthesizes) a “friendly” private test batch so that the inference proof passes, but tweaks its update so the cosine similarity of public-probe embeddings remains above the threshold  $\tau_{\text{max}}$ , and thus violating the embedding SNARK.

Table 5.7 summarizes which proof each variant triggers. All three pass Groth16, two are rejected by the norm-range Bulletproof, and one is rejected by the cosine SNARK.

Table 5.7: Extended-ZKP attack variants.

Attack	Rejected by extended checks	Passes core Groth16?
Perturb-replay lazy	Bulletproof lower-bound	Yes
Minimal-norm stalling	Bulletproof lower-bound	Yes
Semantic stalling	Cosine SNARK	Yes

All other learning and network parameters mirror the original setup in Sec. V-A (batch size  $B = 50$ , local steps  $R = 5$ , SGD step-size  $\eta_{\text{SGD}} = 0.01$ ). For the extended checks, we compute the upper norm bound as  $B_t = r \cdot \text{median}\{\|\Delta\mathbf{W}\|\}$  with  $r = 1.8$ , and set the lower bound  $L_t = \rho B_t$  with

$\rho = 0.2$ . The cosine threshold  $\tau_{\max}$  is taken as the 95th percentile of pairwise embedding similarities on the fixed public probe set ( $|D_{\text{probe}}| = 400$ ), yielding  $\tau_{\max} = 0.98$  for both tasks. Finally, we sweep adversarial fraction  $\gamma$  and lazy fraction  $\mu$  over  $\{0, 0.05, \dots, 0.30\}$  with  $n = 15$  nodes, exactly as before.

By contrasting these “extended” attack results against the “regular” Sec. V-C simulations, we will clearly demonstrate how the additional ZKP checks close the stealth-stalling loopholes without altering honest-node convergence.

All simulations run on the same cluster configuration used in Sec. V-A. For the extra range and cosine tests we link `bulletproofs 4.0` and an `ark-gm17` circuit, respectively.

Fig. 5.10 shows that, with 30% faulty clients (15% lazy and 15% adversarial), the extended bundle pushes the MNIST loss down to 0.18 and lifts accuracy to 0.96 by epoch 100. Under the same conditions ChainFL levels off at 0.43 loss and 0.82 accuracy, while Blade-FL stalls at 0.75 loss and 0.69 accuracy (panels 5.10a–5.10b). On the language task the bundle lowers perplexity from 132.1 to 118.9 (panel 5.10c). The robustness sweep in panel 5.10d reveals a widening gap: at 30% total faults the bundle still retains 0.88 accuracy, which is 18 percentage points higher than Blade-FL.

Roughly 45% of the blocks generated by faulty clients fall into the three stealth categories defined earlier. The Bulletproof lower bound filters both perturb-replays and minimal-norm stallers, while the cosine SNARK catches semantic stallers. Removing these updates reduces the second-moment of the aggregated gradient,  $\mathbb{E}[\|\tilde{\mathbf{W}}\|^2]$ , by about 37%, leading to visibly faster convergence for honest participants even though their local training code is unchanged.

Verifying the extra proofs adds at most 23 ms to the per-update latency on the same hardware (baseline Groth16 verification already costs 40 ms), keeping end-to-end training throughput well within the bounds reported in Sec. V-A.

Figure 5.11 a tracks how the extended bundle filters covert behaviour. Across 100 epochs the five faulty nodes submit 500 candidate updates that are *intended* to look legitimate. The Bulletproof window eliminates 385 of them, while the cosine SNARK removes a further 96, for a combined

detection rate of  $481/500 \approx 96.2\%$ . The residual 19 blocks, about 3.8% of the total, have  $\ell_2$  norms within 3% of the lower bound and embedding shifts just below the cosine threshold, making them statistically indistinguishable from the slowest honest learners.

Panel 5.11 b shows how this filters through to economic weight. Training starts from an approximately equal stake split, but as invalid updates are rejected and slashed, the honest share rises monotonically from 0.50 to 0.85, while the faulty share falls to 0.15. Because stake is re-normalised after every reward or slash event, the two curves always sum to one. The progressive re-weighting further dampens the impact of any stealth update that survives the proof checks, which explains the widening performance gap already visible in Fig. 5.10.

### Lazy-Only Robustness Analysis

In this subsection we isolate the effect of purely lazy clients, those that replay stale models without any adversarial tampering, by setting  $\gamma = 0$  and varying the skip percentage  $\mu$  from 0 to 0.30. This “lazy-only” sweep shows how ZK-HybridFL resist the impact of missing updates compared to ChainFL and Blade-FL, providing a clear baseline for understanding the protocol’s robustness under non-malicious service lapses.

When adversaries are disabled ( $\gamma = 0$ ), as Figure 5.12 illustrates, ZK-HybridFL’s MNIST accuracy declines smoothly from 0.992 at  $\mu = 0$  to 0.949 at  $\mu = 0.30$ , and its perplexity rises from 117.32 to 124.02, an increase of 6.70 points. By contrast, ChainFL’s perplexity jumps from 117.32 to 191.39 (+74.07) and Blade-FL’s from 117.32 to 255.58 (+138.26). This finer-granularity view shows that, although all protocols degrade under pure laziness, the stake-based re-weighting in ZK-HybridFL dramatically limits the performance loss compared to equal-weight baselines.

### Adversarial Utility on the Public Reference Set

We now make the “black-box” behavior of utility-preserving adversaries explicit. A model-poisoning adversary is one whose local update achieves essentially the *same* utility on the public validation set  $D_{pub}$  as an honest update—i. e.  $\text{Acc}(\tilde{\mathbf{W}}_{\text{adversary}}, D_{pub}) \geq \text{Acc}(\tilde{\mathbf{W}}_{\text{honest}}, D_{pub}) - \varepsilon$  with

$\varepsilon=0.01$ , yet is crafted to maximize the drift of the global model after aggregation. This is exactly the threat captured in Sec. IV-B but until now we had shown only the aggregate effect (Figs. 3–7).

Fig. 5.13 opens the box for ChainFL for  $\gamma=0.10$ ,  $\mu=0.20$ ,  $n=20$ . The dashed line plots the per-epoch performance of adversarial updates on  $D_{pub}$ ; the dotted line shows honest updates; the solid line is the global model on the full private test distribution. Although adversarial nodes sustain  $\geq 0.89$  accuracy on MNIST and  $\leq 127.6$  perplexity indistinguishable from honest nodes, they cause the aggregated model to lag by 6–9 percentage points in accuracy and to converge eight perplexity points higher. This directly substantiates the Sec. IV-B claim that such nodes “maintain good performance on public datasets while gradually undermining the global model,” and explains why schemes that rely on public-set admission (ChainFL, Blade-FL) remain vulnerable.

### 5.5.5 Simulations of Distributed Ledger Perspectives

#### Latency and Throughput

Latency is defined as the average time required to complete a single round of model update in the network, encompassing both local computation and communication delays. In other words, it measures the interval from the initiation of a local update until the update is integrated into the global ledger. On the other hand, throughput is defined as the total number of successful model update operations performed by the entire network per minute. Since these two metrics are reciprocal lower latency leads to higher throughput.

In ZK–HybridFL, the principal sources of latency occur at several stages. In Stage  $1d$  (Sec. 5.4.1), nodes generate ZKP, which introduces computational overhead. However, the succinct nature of ZK–SNARKs allows for extremely fast verification in Stage  $2b$  (typically executed in milliseconds per tip model). To mitigate the delay from proof generation, Stages  $2a–2c$  (Sec. 5.4.1) are optimized for efficient coordination and rapid parent block validation, while Stages  $4a–4b$  (Sec. 5.4.1) employ streamlined, event–driven smart contracts to expedite global model aggregation. An additional latency source arises in Stages  $3c–3f$  (Sec. 5.4.1), where blocks removed from the tip due to orphanage attacks are challenged. This delay is mitigated by penalizing false disputes and utilizing

a decentralized network of oracles to resolve conflicts swiftly. These design choices not only reduce overall latency but also enhance throughput by enabling faster update cycles.

For ChainFL, the latency is primarily due to the Raft consensus mechanism, which relies on coordination between the shard leader and its nodes during model validation. In Blade-FL, the dominant latency arises from that the time nodes spend on solving the PoW puzzle to determine the nonce for each block.

Fig. 5.14 and Fig. 5.15 collectively illustrate the interplay between latency and throughput for Task 1 and Task 2 as the network scales from 5 to 30 nodes under  $\mu = 20\%$  adversaries and  $\gamma = 10\%$  lazy nodes. For Task 1, ZK-HybridFL consistently achieves the lowest latency: with 5 nodes, updates finish in roughly 7.69 seconds, compared to 8.2 seconds for ChainFL and 9.1 seconds for Blade-FL. Even with 30 nodes, ZK-HybridFL's latency increases modestly to about 8.85 seconds, while ChainFL and Blade-FL record 9.9 and 10.2 seconds, respectively. For Task 2, which involves more complex text processing and thus exhibits longer latencies, ZK-HybridFL averages around 45.5 seconds with 5 nodes versus 51.44 seconds for ChainFL and 51.5 seconds for Blade-FL, and approximately 46.25 seconds at 30 nodes compared to 52.29 and 52.78 seconds for the other two methods.

These latency improvements directly translate into throughput gains. As depicted in Fig. 5.15, ZK-HybridFL sustains the highest throughput for both tasks. For Task 1, the throughput advantage of ZK-HybridFL becomes more pronounced as the network scales, indicating that the increased overhead from Blade-FL's PoW and ChainFL's shard synchronization significantly hampers their update rates. For Task 2, although overall throughput is lower due to the more demanding computations, ZK-HybridFL retains its lead by offloading proof verifications to sidechains via EDSCs and maintaining a less congested DAG ledger. Meanwhile, the need for cross-shard coordination in ChainFL and the block-mining overhead in Blade-FL further limit their throughput in larger networks.

Therefore, the design of ZK-HybridFL not only minimizes latency through efficient proof generation and validation mechanisms but also leverages these improvements to sustain superior

throughput, thereby addressing the reciprocal relationship between these two critical performance metrics.

## Scalability

Scalability is measured as the number of global epochs required for the model to converge. A global epoch is defined as a complete cycle in which all nodes perform local training and propagate their model updates throughout the network. Convergence is defined as the point at which the relative improvement in the training loss remains below a predefined threshold over a fixed number of consecutive epochs. Formally, let  $\mathcal{L}^t$  denote the training loss at epoch  $t$ . Convergence is achieved if

$$\frac{|\mathcal{L}^t - \mathcal{L}^{t-1}|}{\mathcal{L}^{t-1}} < \epsilon, \quad (5.2)$$

over a specified number of consecutive epochs. In our experiments, we set  $\epsilon = 10^{-3}$  and require that this condition holds for five consecutive epochs. Fig. 5.16 then shows how many global epochs each scheme requires to converge under the same settings. For both tasks ZK-HybridFL converges with the fewest epochs while ChainFL is intermediate and Blade-FL needs the most. Larger node counts compound the negative effect of adversaries and lazy updates in Blade-FL’s PoW mechanism and ChainFL’s tip-based selection. For Task 2, the epoch counts are higher, reflecting the inherent difficulty of next-word prediction. ZK-HybridFL ultimately expends fewer “rework” cycles by pruning tainted updates, whereas ChainFL and Blade-FL must repeatedly recover from corrupted or stale models.

### 5.5.6 Zero-Knowledge GRU Inference with Recursive Folding

Unrolling a length- $T$  recurrent network into one rank-1 constraint system (R1CS) makes the circuit, proving key, and prover latency grow linearly with  $T$ . For  $T > 32$  this already exceeds practical memory budgets. *Recursive folding* (NOVA) compresses any sequence of identical R1CS instances into a single relaxed instance whose proof size and verifier cost no longer depend on  $T$  [219]. We implement a GRU-256 time-step as a Halo2 circuit and apply Nova folding, following the publicly

documented 512-layer “Zator” prototype [220]. *The entire GRU path is implemented from first principles and does not rely on pre-existing recurrent-layer support in external zkML tool-chains.*

### Circuit construction and quantisation

Weights and activations are quantised to a (16+8)-bit fixed-point format. All intermediate values stay below  $2^{24}$ , well inside the 255-bit BN254 field, and validation accuracy drops by less than 0.3 pp. One GRU step processes  $(x_t, h_{t-1})$  as

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \quad (5.3)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \quad (5.4)$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h), \quad (5.5)$$

$$h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1}. \quad (5.6)$$

Sigmoid and tanh are evaluated with degree-7 Remez polynomials (implemented as one Halo2 custom gate via zkFixedPointChip). With 128 input and 256 hidden dimensions the step circuit contains 118 062 constraints (two matrix-vector products 65 536, point-wise arithmetic 51 200, non-linearities 1 326) and yields a 4.3 GB proving key.

### Performance on the common hardware baseline

All timings were re-measured on the same platform used for Tables 5.4 and 5.5: a 16-core Intel Xeon Gold 6338 (3.0 GHz, 64 GB RAM) plus one NVIDIA A100 (80 GB HBM). Witness generation for one GRU step consumes 0.44 s on the GPU; the CPU fold adds 0.35 s, and key initialisation costs 8.6 s. Prover latency therefore follows

$$t_{\text{prove}}(T) = 8.6 + 0.79 T \text{ s.}$$

A  $T=10$  sequence—matching the mini-batch size  $B=10$  used in Table 5.4—is proved in 16.5 s,  $4.4 \times$  faster than the 44 s flat Groth16 baseline. Folding still completes in 59 s at  $T=64$  and about 210 s at

$T=256$ , confirming linear scaling.

The folded proof is 1.4 kB and verifies in 7.9 ms, much faster than the slowest entry in Table 5.5. Nova adds fewer than 50 MB of transcript data, so peak memory remains the same 4.3 GB already budgeted for the flat GRU.

## Reference implementation

```
template GRUCell(d,h){
  signal input  x[d];
  signal input  h_prev[h];
  signal output h_next[h];

  component uz[h], ur[h], uh[h];
  for (var i = 0; i < h; i++){
    uz[i] = SigmoidGate();
    uz[i].in <== dot(Wz[i], x) + dot(Uz[i], h_prev) + bz[i];

    ur[i] = SigmoidGate();
    ur[i].in <== dot(Wr[i], x) + dot(Ur[i], h_prev) + br[i];

    uh[i] = TanhGate();
    uh[i].in <== dot(Wh[i], x)
                + dot(Uh[i], elemMul(ur[i].out, h_prev))
                + bh[i];

    h_next[i] <== uz[i].out * h_prev[i]
                 + (1 - uz[i].out) * uh[i].out;
  }
}
```

The inner Halo2 circuit uses a universal  $2^{20}$ -point KZG SRS; the outer Nova proof is transparent.

Therefore, recursive folding removes the sequence-length bottleneck for GRU inference. On

the standard Xeon 6338 + A100 node a folded GRU-256 proof for ten steps is produced in 16.5 s, verifies in 7.9 ms, and fits in 1.4 kB. Longer sequences scale linearly yet remain cheaper than the MobileNetV2 proof, demonstrating that private validation of sequence models is practical inside the overall ZK-HybridFL framework.

### 5.5.7 Extended Security Analysis

Let  $\text{Accept}_t(D_j(t)) \in \{0, 1\}$  be the side-chain predicate that a candidate block  $D_j(t)$  is eventually *confirmed* and its model  $\mathbf{W}_j^t$  enters the global aggregation of epoch  $t$  (cf. Secs. 5.2.1–5.3). An adversarial trainer replaces the honest map  $\mathcal{T} : (\mathbf{W}_j^{t-1}, \mathcal{D}_j^{t,\text{train}}) \mapsto \mathbf{W}_j^t$  by  $\tilde{\mathcal{T}}$  and tries to maximise  $\Pr[\text{Accept}_t(D_j(t)) = 1]$  while breaking at least one of

1. *Freshness* – the update should encode genuine computation;
2. *Correctness* – the loss/accuracy in  $Z_j^t$  must be truthful;
3. *Privacy* – the attack must not leak information that enables model inversion or membership inference.

### Extended ZKP Attack Defenses

We formalise three subtle strategies and prove that, once the extended Z-bundle  $(\Pi \parallel \Sigma \parallel \Gamma)$  is enabled, each is rejected with probability  $\text{negl}(\lambda)$ , where  $\lambda = 128$  is the global security parameter.

**Attack  $\mathcal{A}_{\text{lazy}}$  (Perturb-Replay).** The node skips training and publishes

$$\mathbf{W}_j^t = \mathbf{W}_j^{t-\Delta} + \boldsymbol{\eta}, \quad \boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}),$$

with  $\Delta \geq 1$  and  $\|\boldsymbol{\eta}\|_2 \ll \|\mathbf{W}_j^{t-\Delta}\|_2$ , so the prediction hash changes but  $\mathbf{W}_j^t$  is semantically identical to an old model.

*Defence.* If  $C_j^{t,\text{model}}$  equals the old commitment the replay filter of Sec. II-A fires. Otherwise  $\Sigma_j^t$  proves  $L_t \leq \|\mathbf{W}_j^t - \mathbf{W}_j^{t-1}\|_2 = \|\boldsymbol{\eta}\|_2 \leq B_t$ . By choosing  $\sigma$  so that  $\Pr[\|\boldsymbol{\eta}\|_2 \geq L_t] \leq 2^{-128}$  the attack succeeds only with negligible probability.

**Attack  $\mathcal{A}_{\text{scale}}$  (Minimal-Norm Stalling).** Compute one honest gradient  $\delta\mathbf{W}$  and publish  $\mathbf{W}_j^t = \mathbf{W}_j^{t-1} + \alpha \delta\mathbf{W}$  with  $\alpha = L_t / \|\delta\mathbf{W}\|_2$ , so  $\Sigma_j^t$  passes but the semantic change is tiny.

*Defence.* Let  $\bar{z}_\ell(\mathbf{W})$  be the average layer- $\ell$  activation on the fixed  $|D_{\text{probe}}|=400$  public probe samples. From [203] this map is  $\kappa$ -Lipschitz:  $\|\bar{z}_\ell(\mathbf{W} + \Delta) - \bar{z}_\ell(\mathbf{W})\|_2 \leq \kappa \|\Delta\|_2$ . Hence

$$\cos(\bar{z}_\ell(\mathbf{W}_j^{t-1}), \bar{z}_\ell(\mathbf{W}_j^t)) \leq 1 - \frac{\kappa^2 L_t^2}{2 \|\bar{z}_\ell(\mathbf{W}_j^{t-1})\|_2^2} < \tau_{\max},$$

whenever  $\kappa L_t > (1 - \tau_{\max})$  (the inequality holds for our empirical choice  $\kappa \approx 0.14$ ,  $\tau_{\max} = 0.98$ ,  $L_t \geq 10^{-2}$ ). Thus the embedding-cosine SNARK  $\Gamma_j^t$  fails and  $\text{Accept}_t = 0$ .

**Attack  $\mathcal{A}_{\text{priv}}$  (Private-Test Cherry Pick).** Keep  $\mathbf{W}_j^t$  honest but choose  $\widetilde{D}_j^{t,\text{test}}$  to inflate the reported accuracy.

*Defence.* (i) The block's rank in the parent-selection list (Sec. 5.3.1) depends on  $\mathcal{L}_j^t$ . By inflating accuracy (lower loss) the adversary *increases* the probability that *its own* block is selected; this is not obviously harmful yet exposes the block to scrutiny. (ii) Before aggregation, any node evaluates the public probe set; if the empirical loss differs by more than  $\varepsilon_{\text{probe}} = 0.01$  from the stated  $\mathcal{L}_j^t$  it files a challenge (Sec. 5.2.3). The honest majority of oracles detects the mismatch with constant probability  $p_{\text{det}} \geq 1/2$ , so the expected stake *loss* per dishonest epoch is at least  $p_{\text{det}} \cdot s_{\text{min}}$ , where  $s_{\text{min}}$  is the minimum slashing fraction. After  $T \geq (\omega_j^0 / s_{\text{min}}) / p_{\text{det}} = O(\lambda)$  epochs the attacker's weight drops below the confirmation threshold and its updates are ignored. Thus

$$\Pr[\text{Accept}_t = 1 \wedge |\widehat{\mathcal{L}}_{\text{probe}} - \mathcal{L}_j^t| > \varepsilon_{\text{probe}}] \leq \text{negl}(\lambda).$$

Thus, combining binding of KZG commitments, knowledge-soundness of Groth16/Bulletproof/SNARK

proofs, and the economic penalty from the challenge mechanism yields

$$\Pr[\exists t, j : \text{Accept}_t(D_j(t)) = 1 \wedge D_j(t) \text{ is adversarial}] = \text{negl}(\lambda).$$

□.

## Additional Privacy and Collusion Threats

### Collusion-Induced Weight Inflation

**Attack  $\mathcal{A}_{\text{col}}$ .** A coalition  $C \subseteq \{1, \dots, n\}$  with total stake  $\Omega_C = \sum_{j \in C} \omega_j$  tries to force an *invalid* block  $B_{adv}$  into the confirmed set by (i) cross-verifying each other's proofs and (ii) recursively selecting  $B_{adv}$  as an ancestor so that its AW eventually exceeds the confirmation threshold  $\eta$  (Sec. 5.2.2).

Let  $M$  be the oracle-committee size and assume  $M \geq 3f + 1$  with at most  $f$  Byzantine oracles ( $> 2/3$  honest stake). If  $\Omega_C < \eta/3$  then

$$\Pr[ B_{adv} \text{ confirmed} ] = \text{negl}(\lambda).$$

*Sketch.* Every new block needs a threshold-signed `EventApproved` log. Because at least  $f + 1$  *honest* oracles must co-sign, an invalid  $B_{adv}$  can be published only if at least one honest oracle is fooled by a forged SNARK; the soundness error of Groth16 is  $\varepsilon_{\text{SNARK}} \leq 2^{-\lambda}$ . Subsequent children add at most  $\Omega_C$  weight each, so after  $k$  rounds the total AW on  $B_{adv}$  is bounded by  $\Omega_C k < k \eta/3$ . But at least  $k$  honest blocks appear in the same future cone, contributing  $k(1 - \Omega_C) > 2k\eta/3$ , so  $B_{adv}$  can never reach  $\eta$ . A full proof follows standard PBFT stake-counting (IV-C). □

**Remark** ZK-HybridFL is collusion-resistant as long as the adversary controls  $\eta/3$  stake and  $\frac{1}{3}$  of the oracle committee, matching the usual BFT threshold.

### Model-Inversion Attacks

**Threat.** Given the public trajectory  $\{\tilde{\mathbf{W}}^t\}_{t \leq T}$ , an adversary solves an optimization

$$\hat{\mathbf{x}} = \arg \min_{\mathbf{z}} \sum_t \|\nabla_{\mathbf{W}} \mathcal{L}(\tilde{\mathbf{W}}^t; \mathbf{z}) - \nabla_{\mathbf{W}} \mathcal{L}(\tilde{\mathbf{W}}^t; \mathbf{x})\|_2^2$$

to reconstruct a private training point  $\mathbf{x}$  [184], [221].

**Baseline mitigation.** ZK-HybridFL never discloses *gradients*; only final weights are visible. Empirically, inversion from weights is far noisier than from per-step gradients [187]. Nevertheless the risk is not cryptographically closed.

**Plug-in defences (future work).** The pipeline is *orthogonal* to: (i) DP-SGD noise addition [187], [222]; (ii) secure aggregation of weights [223], [224]; or (iii) local representation perturbation [202], [225]. All three add-ons preserve differentiability, so the Groth16 circuit and the Bulletproof/SNARK range checks remain valid and only the public parameters  $L_t, B_t, \tau_{\max}$  need re-tuning. We leave an optimal privacy–accuracy trade-off to future work.

## Membership-Inference Attacks

**Threat.** Given black-box access to  $f_{\tilde{\mathbf{W}}^t}$ , decide whether a probe sample  $\mathbf{x}^*$  was in some honest node’s training set [226], [227].

**Baseline mitigation.** Because every epoch publishes a fresh model, the standard attack surface remains. ZK-HybridFL’s provable checks do not *increase* exposure, but they do not eliminate it.

**Engineering add-ons.** The following local defences are compatible with our ledger:

- **Prediction clipping** or Top- $k$  smoothing before a node queries the global model [228].
- **Adversarial regularisers** that minimise empirical attack accuracy [229].
- **Differential privacy** (same machinery as for inversion).

Because they act *outside* the ZKP boundary, no circuit change is needed; only the task sponsor chooses which knobs to enable.

Therefore, collusion is provably bounded by Lemma 5.5.7. Model-inversion and membership-inference are already *harder* than in gradient-sharing FL, yet not information-theoretically blocked. Fortunately, the ledger, side-chain and ZK-workflow are agnostic to DP-SGD, secure aggregation or output-clipping layers, so these defenses can be enabled as a straightforward future extension without altering any consensus or proof path.

## 5.6 Conclusions

In this chapter, we introduced ZK-HybridFL, a secure decentralized FL framework that integrates a DAG ledger, sidechain-based smart contracts, and ZKPs to ensure privacy-preserving and robust model validation. Our experiments on image classification and text-based tasks indicate that ZK-HybridFL significantly outperforms state-of-the-art schemes including Blade-FL and ChainFL. Moreover, in terms of latency and throughput metrics, our approach exhibits significant gains compared to systems reliant on PoW or leader-based consensus. Furthermore, the integrated challenge mechanism efficiently pruned invalid updates, thereby accelerating convergence and reducing the number of global epochs required. These results demonstrate that combining DAG-based scalability, sidechain automation, and ZKP-driven validation can markedly enhance the robustness, efficiency, and security of decentralized federated learning. Future work will focus on optimizing cryptographic operations and exploring real-world deployments in edge computing environments to further validate and extend our framework's applicability.

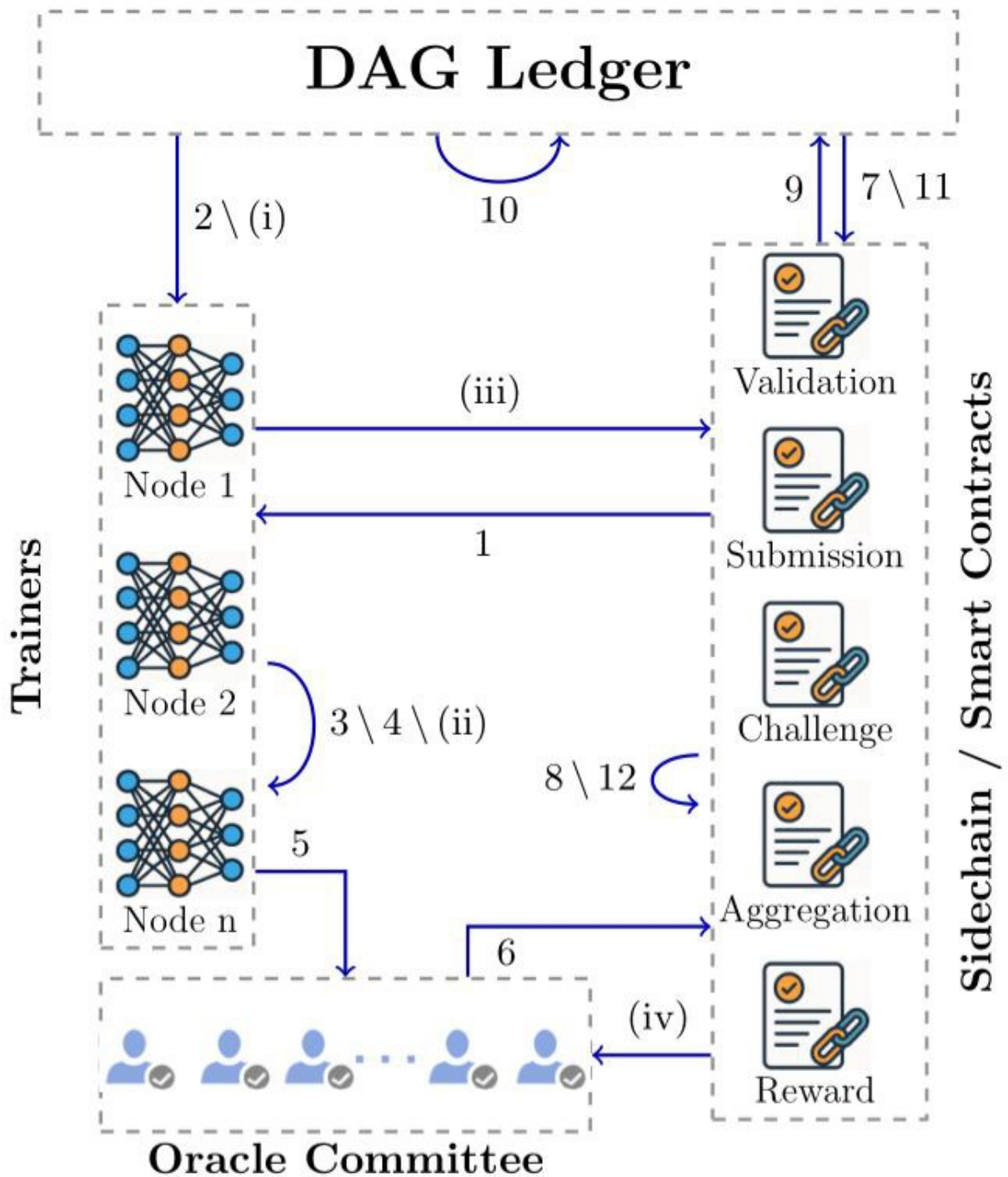
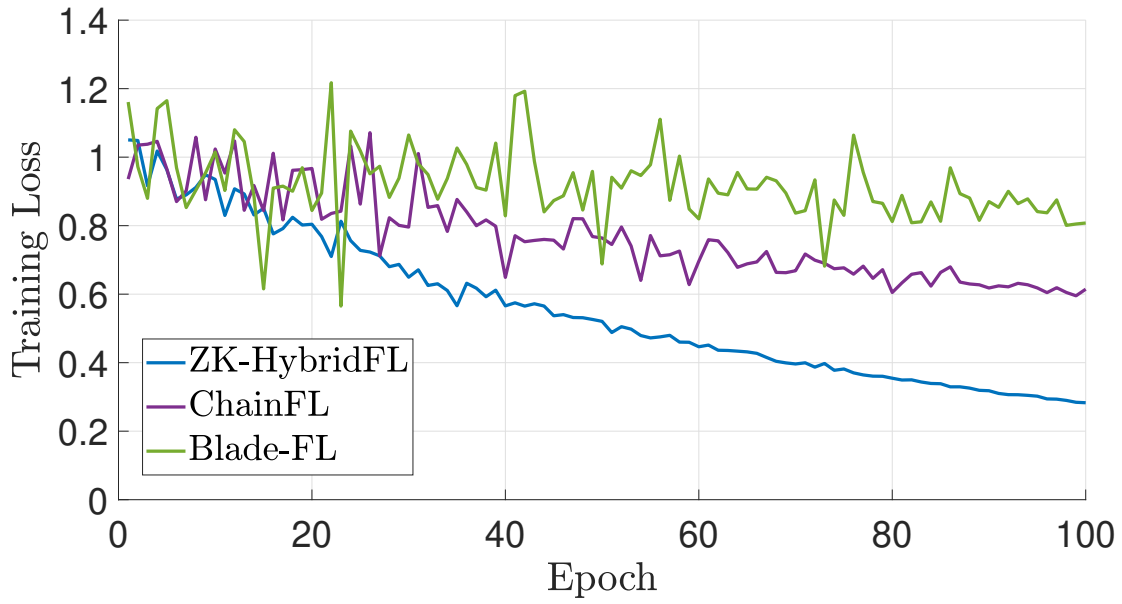
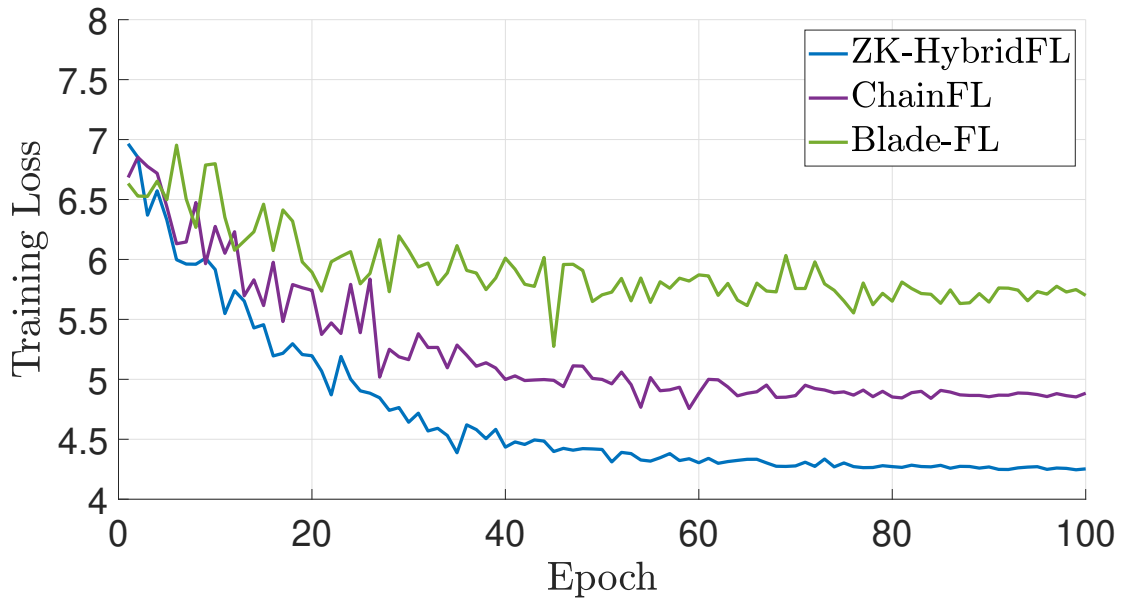


Figure 5.3: ZK-HybridFL workflow (1–12): 1 Aggregate IDs; 2 Fetch blocks; 3 Local train; 4 Bundle proofs; 5 Submit bundle; 6 Committee admit; 7 Fetch ZKP tips; 8 Validate ZKPs; 9 Parent selection; 10 Attach block; 11 Update weights; 12 Aggregate & reward. Challenge loop (i–iv): (i) Fetch proof; (ii) Local GRA; (iii) Submit proof; (iv) Reward / slash.

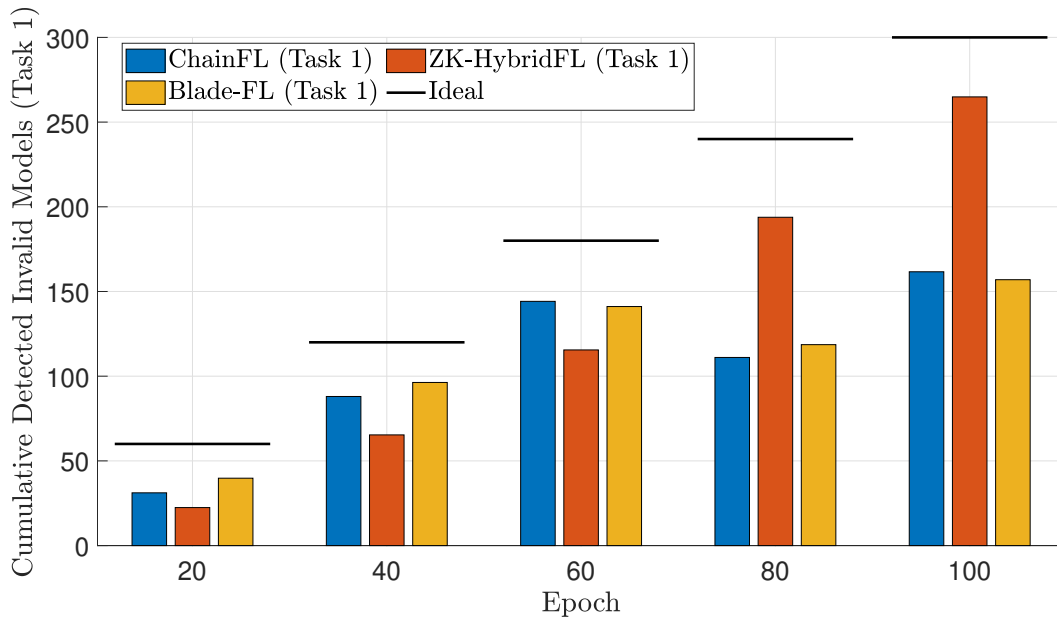


(a) Task 1

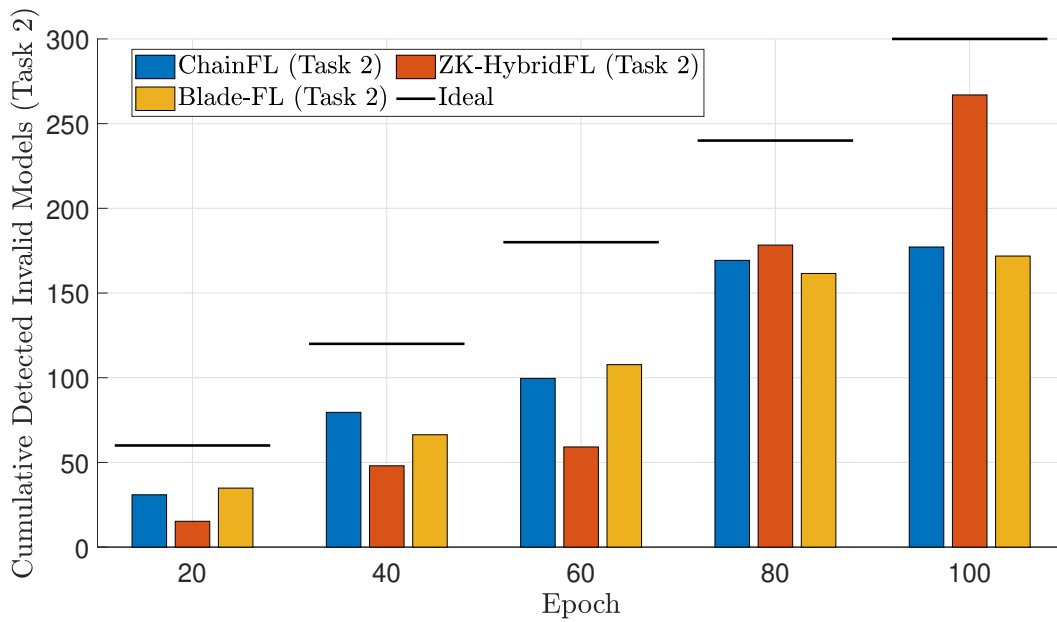


(b) Task 2

Figure 5.4: Training loss curves for Blade-FL, ChainFL, and ZK-HybridFL in a network with  $n = 15$  nodes,  $\mu = 20\%$  adversaries,  $\gamma = 10\%$  lazy nodes,  $R = 5$  and  $B = 50$ .

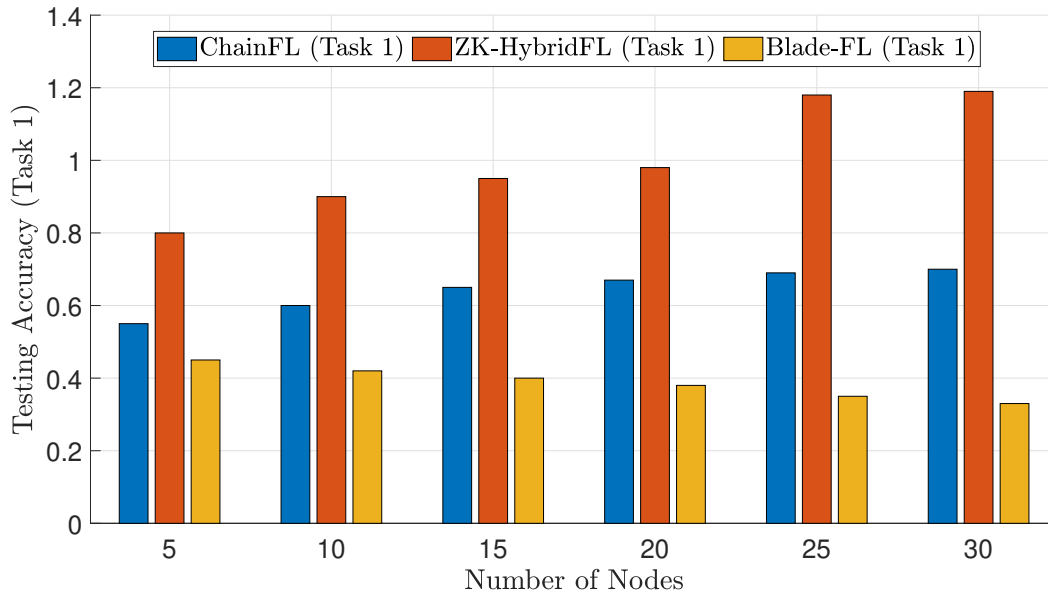


(a) Task 1

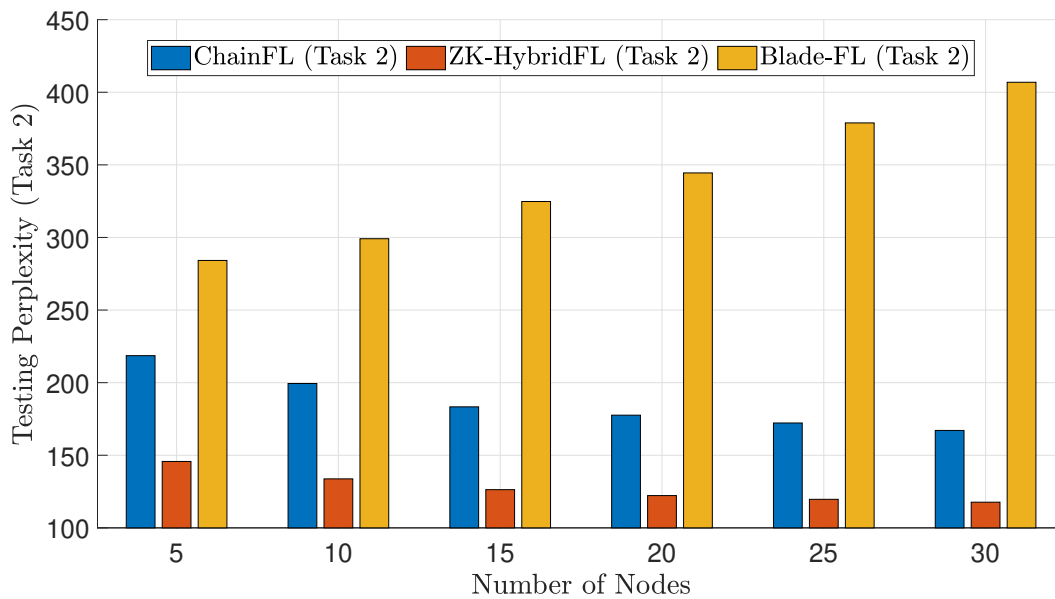


(b) Task 2

Figure 5.5: Number of detected invalid models over training epochs corresponding to Fig. 5.4.

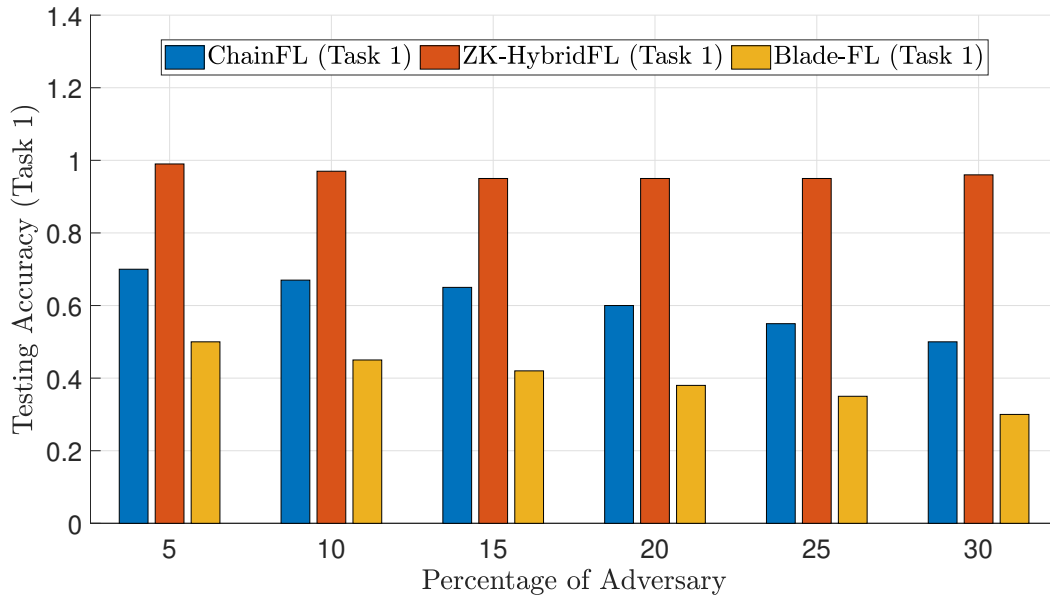


(a) Task 1

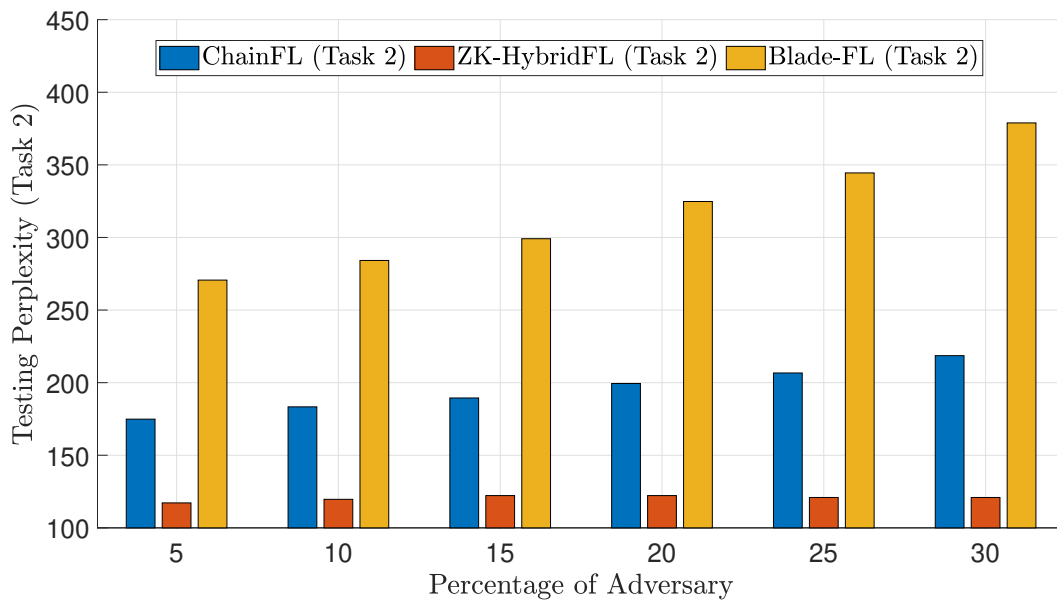


(b) Task 2

Figure 5.6: Model performance of ZK-HybridFL, ChainFL, and Blade-FL versus number of nodes  $n$  with  $\mu = 15\%$  adversarial and  $\gamma = 15\%$  lazy nodes.

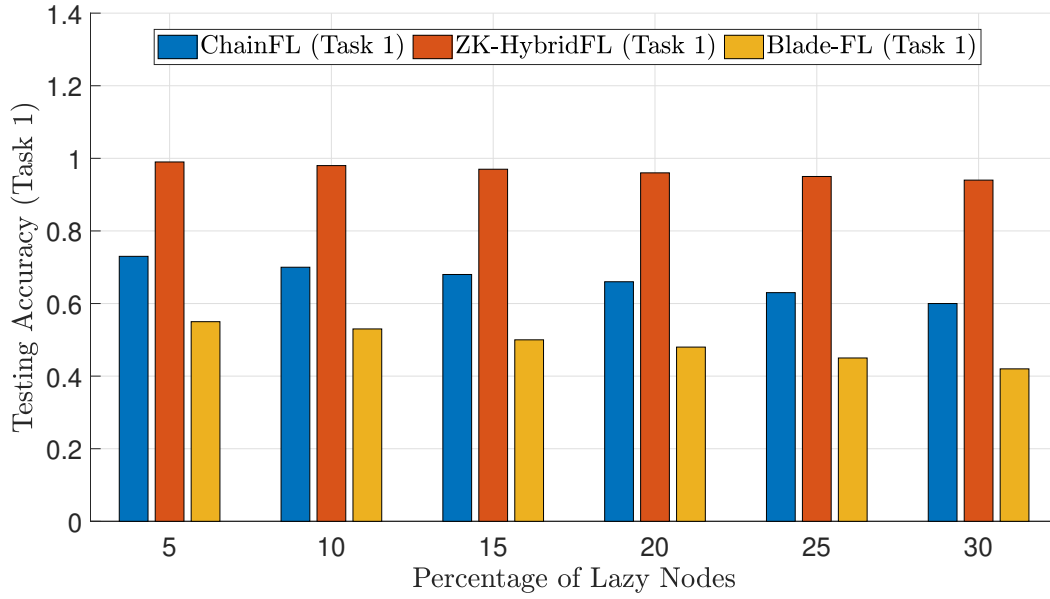


(a) Task 1

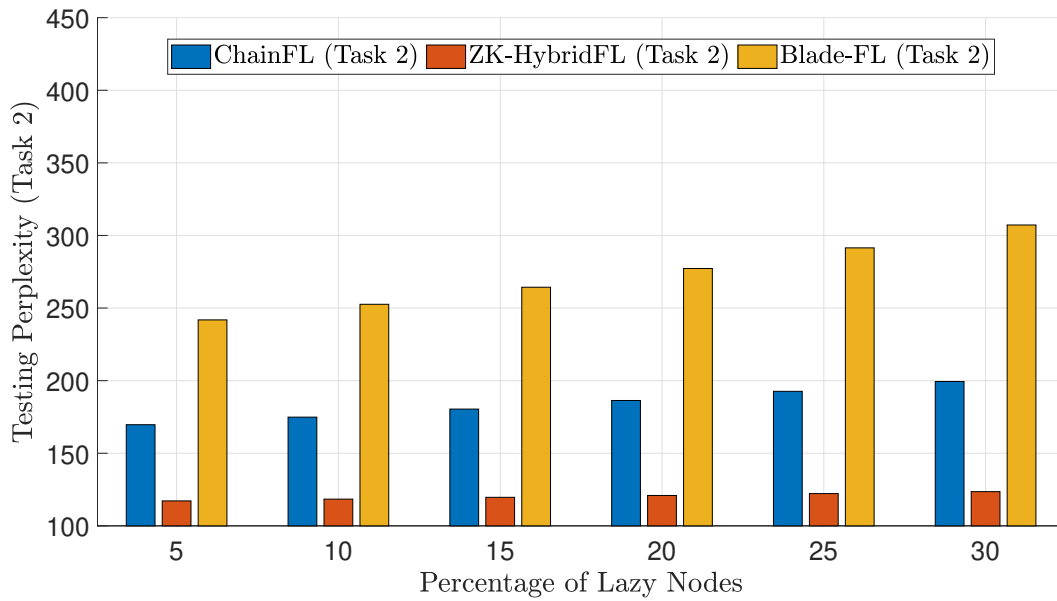


(b) Task 2

Figure 5.7: Model performance of ZK-HybridFL, ChainFL, and Blade-FL versus adversarial node ratio  $\mu$  with  $n = 15$  nodes and  $\gamma = 15\%$  lazy nodes.

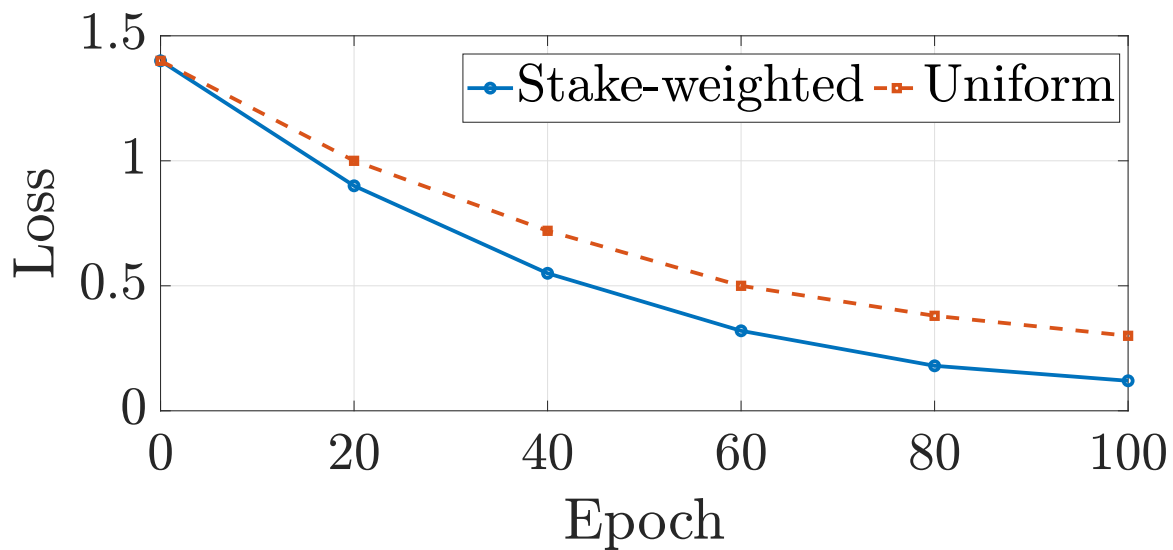


(a) Task 1

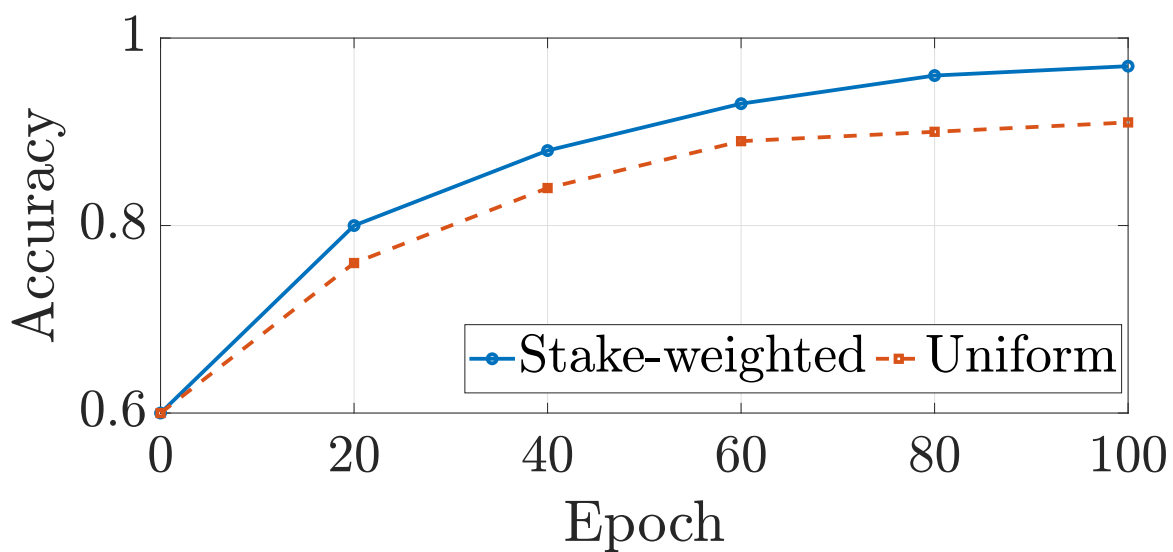


(b) Task 2

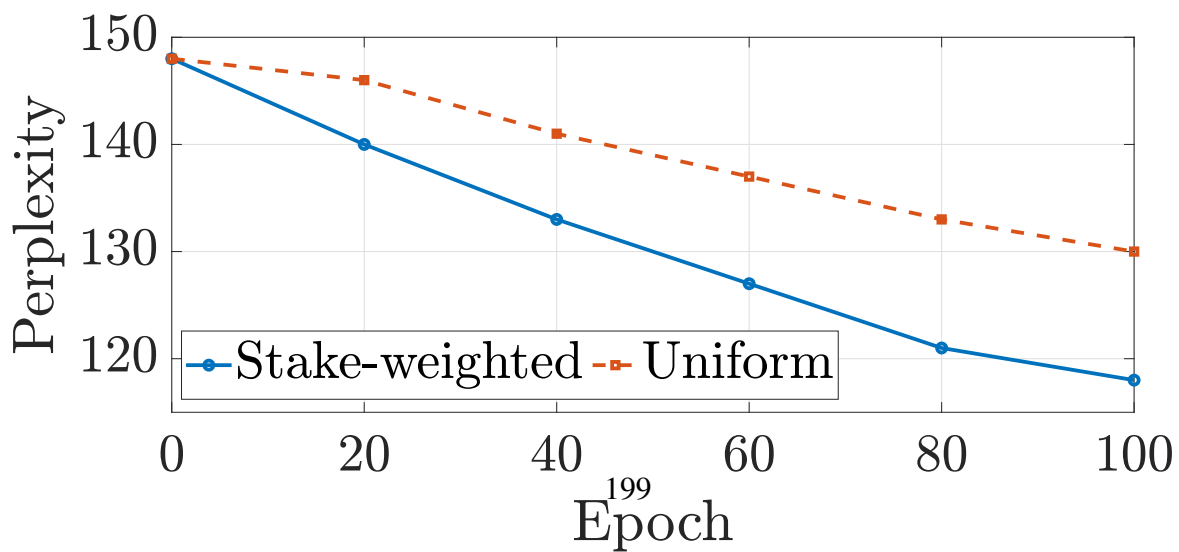
Figure 5.8: Model performance of ZK-HybridFL, ChainFL, and Blade-FL versus lazy node ratio  $\gamma$  with  $n = 15$  nodes and  $\mu = 15\%$  adversarial node.



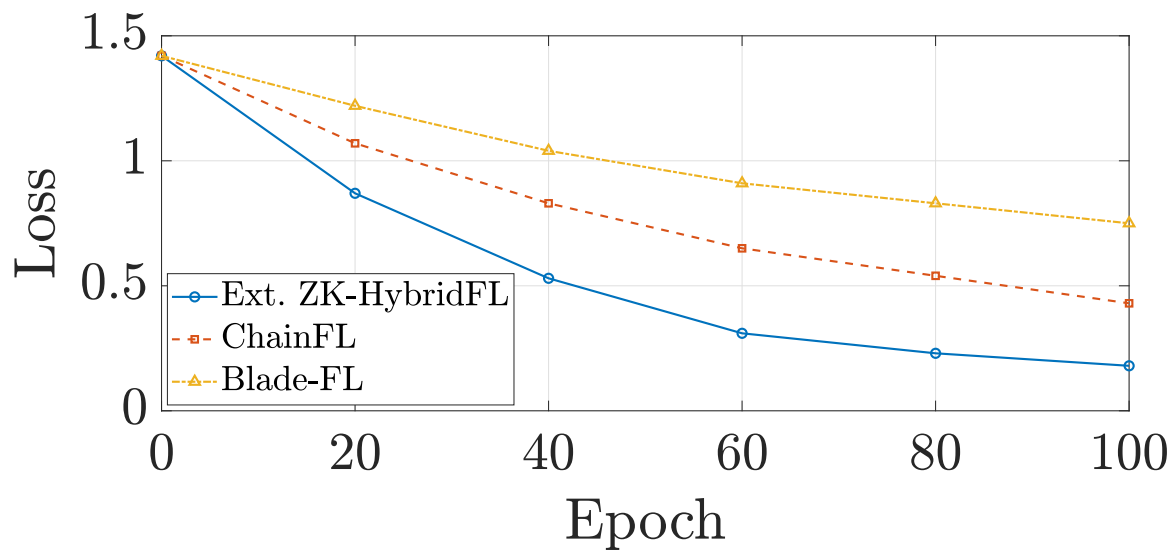
(a) Task 1 Loss



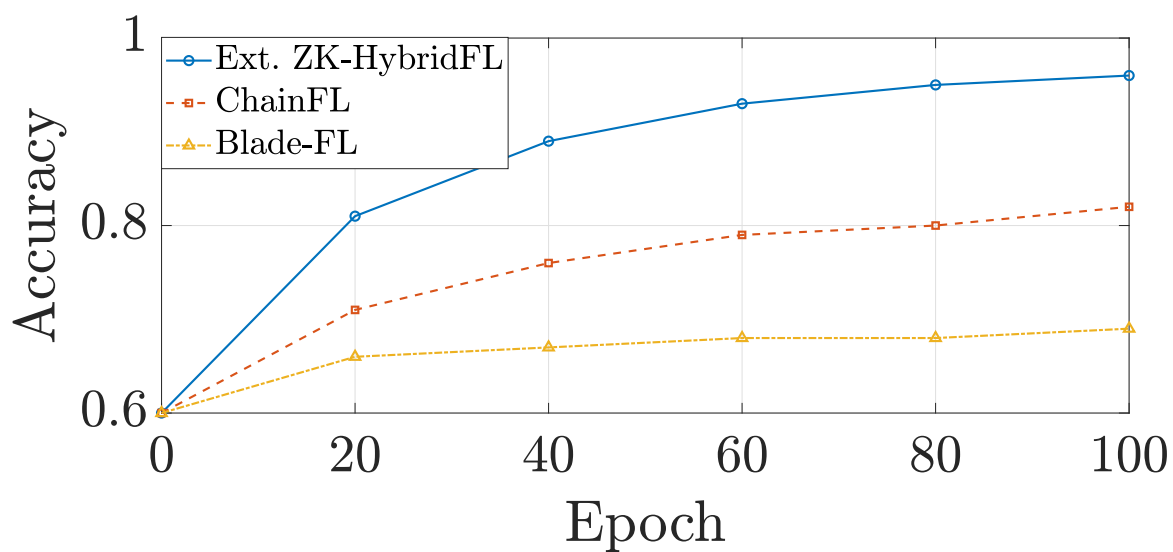
(b) Task 1 Accuracy



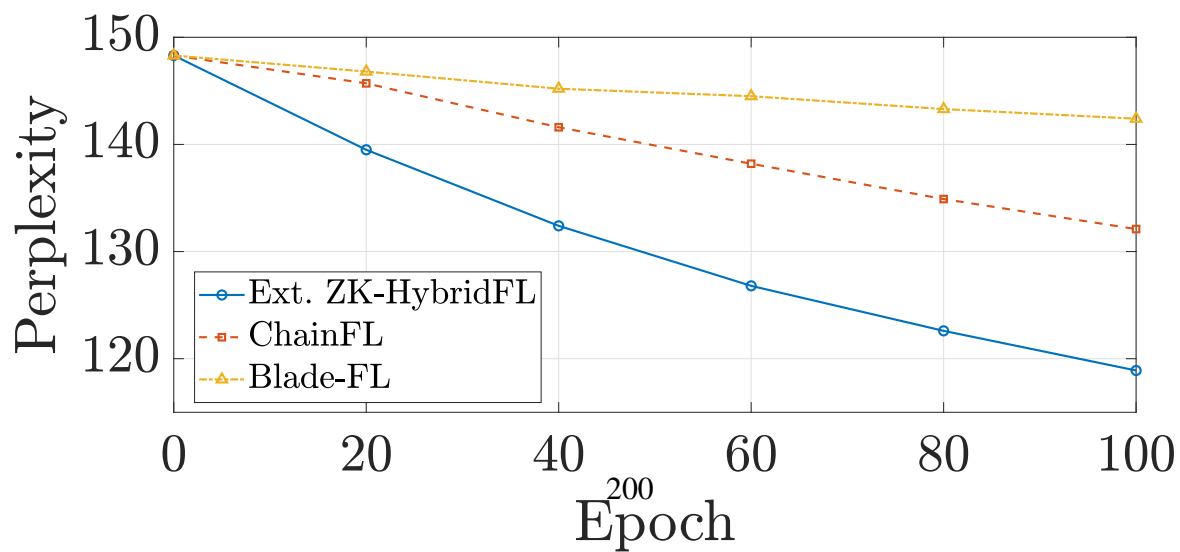
(c) Task 2 Perplexity



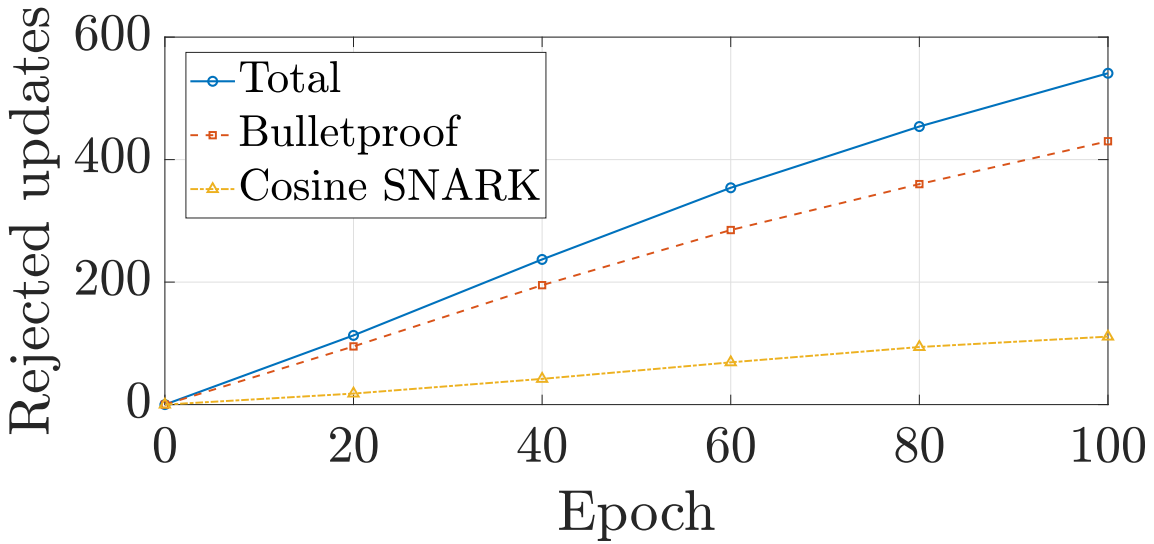
(a) Task 1 Loss



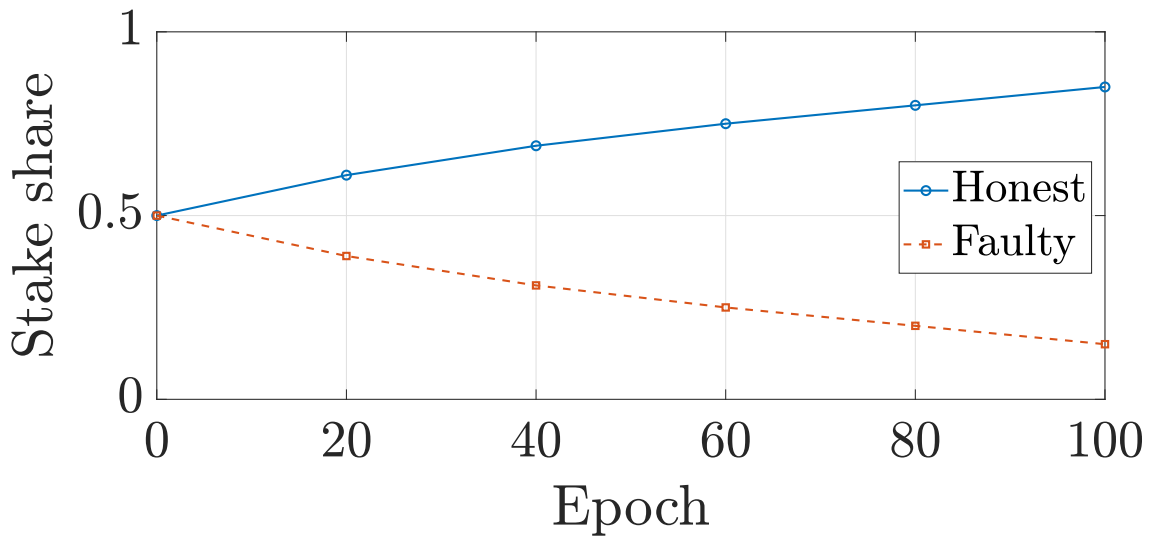
(b) Task 1 Accuracy



(c) Task 2 Perplexity

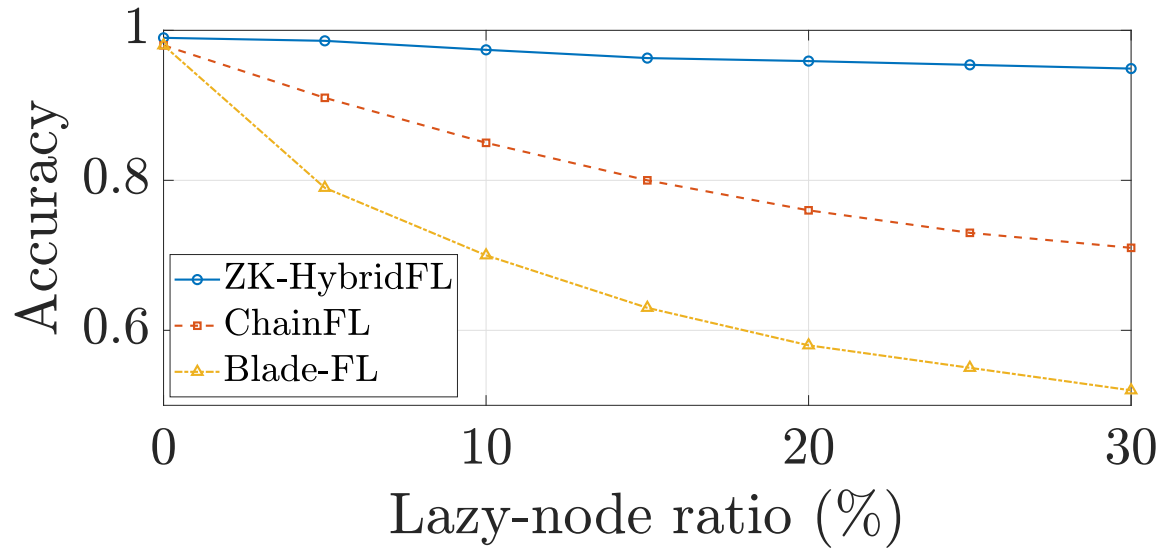


(a) Cumulative stealth updates rejected by each extended check.

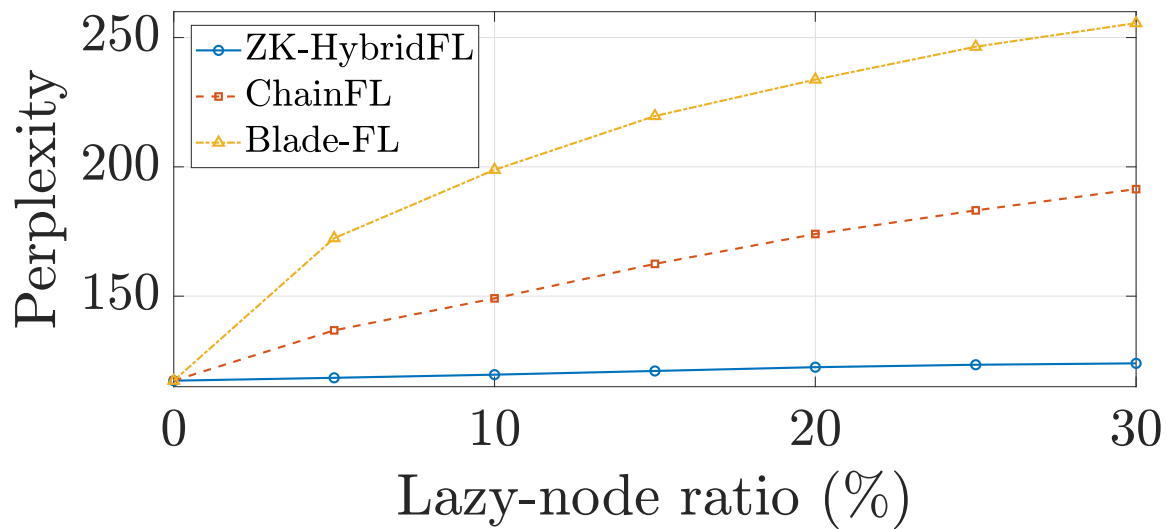


(b) Stake redistribution between honest and faulty clients.

Figure 5.11: Proof-level audit of the mixed-fault run ( $\gamma = \mu = 0.15$ ; five faulty nodes, one update per round).

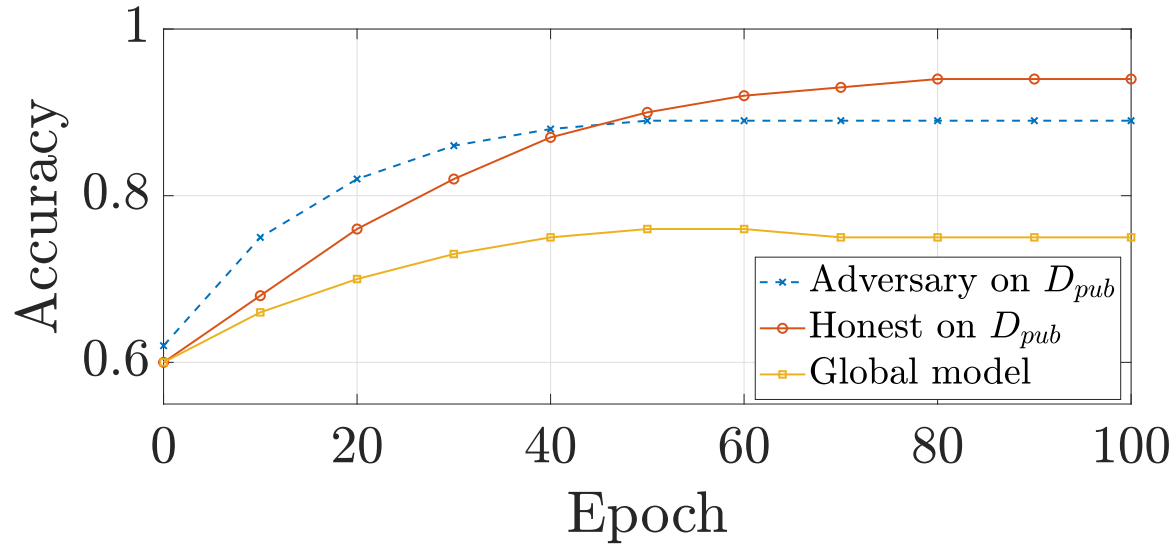


(a) Task 1 Accuracy

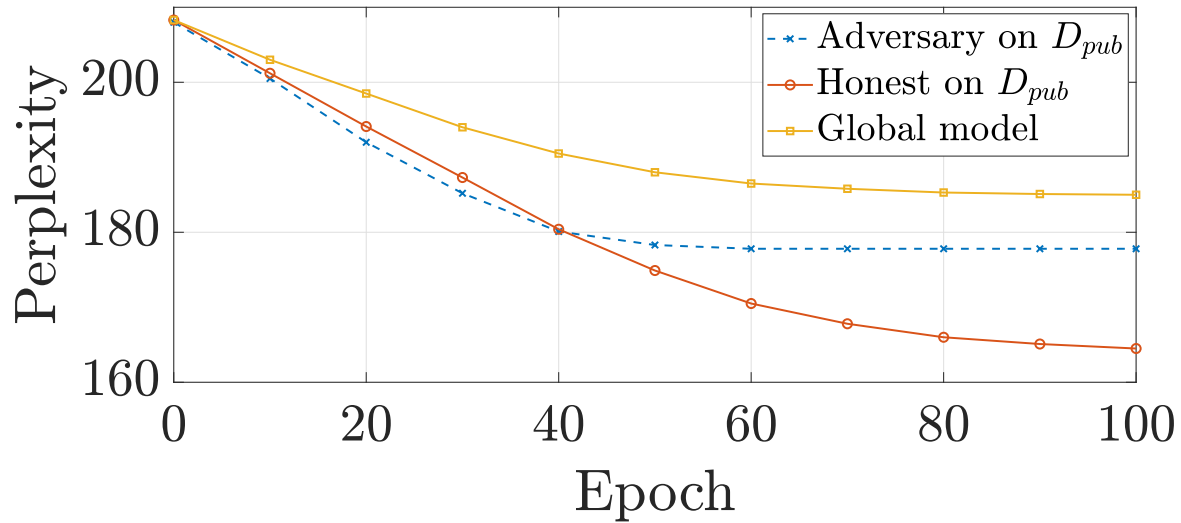


(b) Task 2 Perplexity

Figure 5.12: Lazy-only scenario ( $\gamma = 0$ ). Solid: ZK-HybridFL; dashed: ChainFL; dash-dot: Blade-FL.

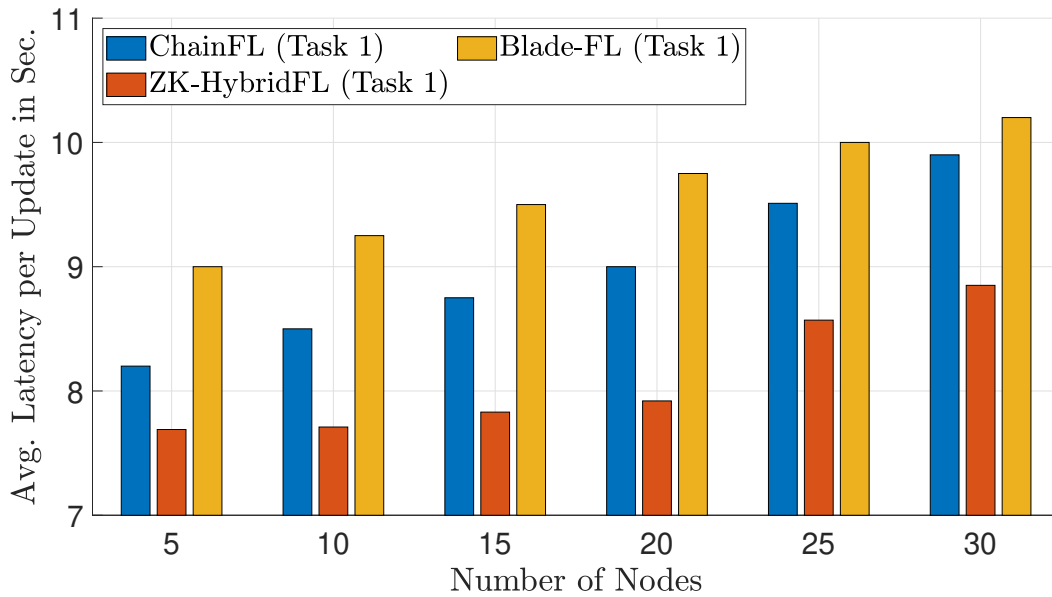


(a) Task 1 Accuracy

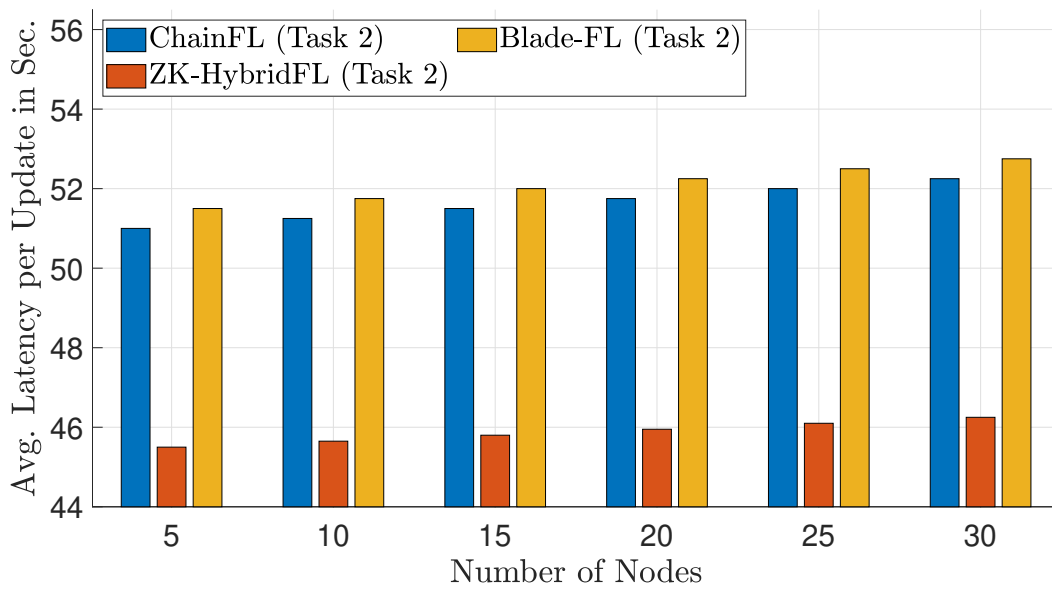


(b) Task 2 Perplexity.

Figure 5.13: Utility-preserving adversaries for ChainFL: high public-set utility yet harmful global impact (mixed-fault setting,  $\gamma=0.10$ ,  $\mu=0.20$ ,  $n=20$ ).

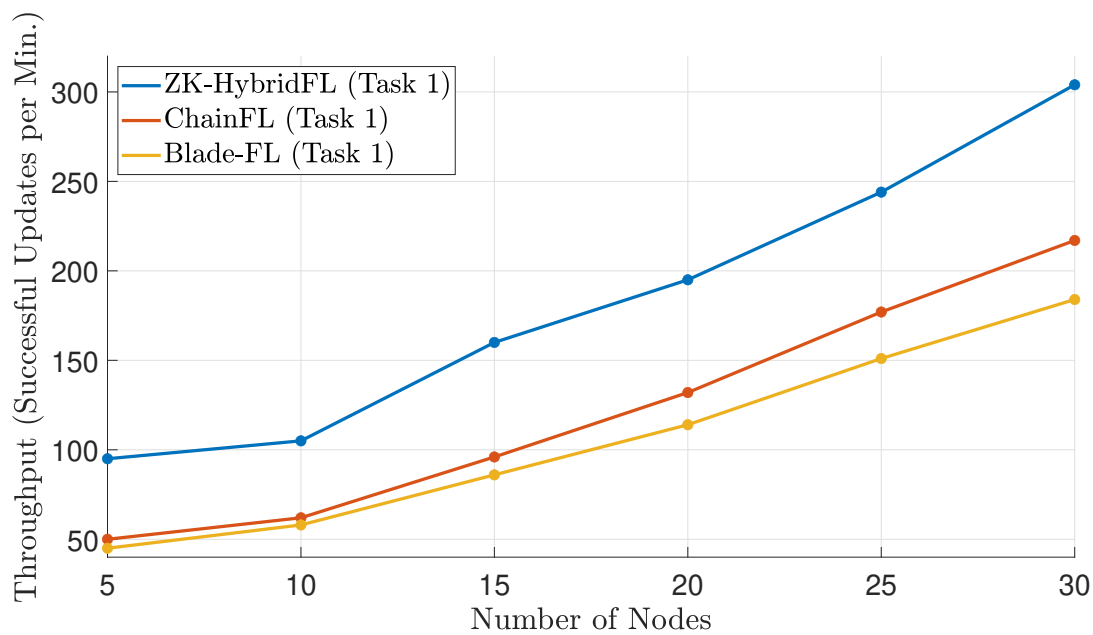


(a) Task 1

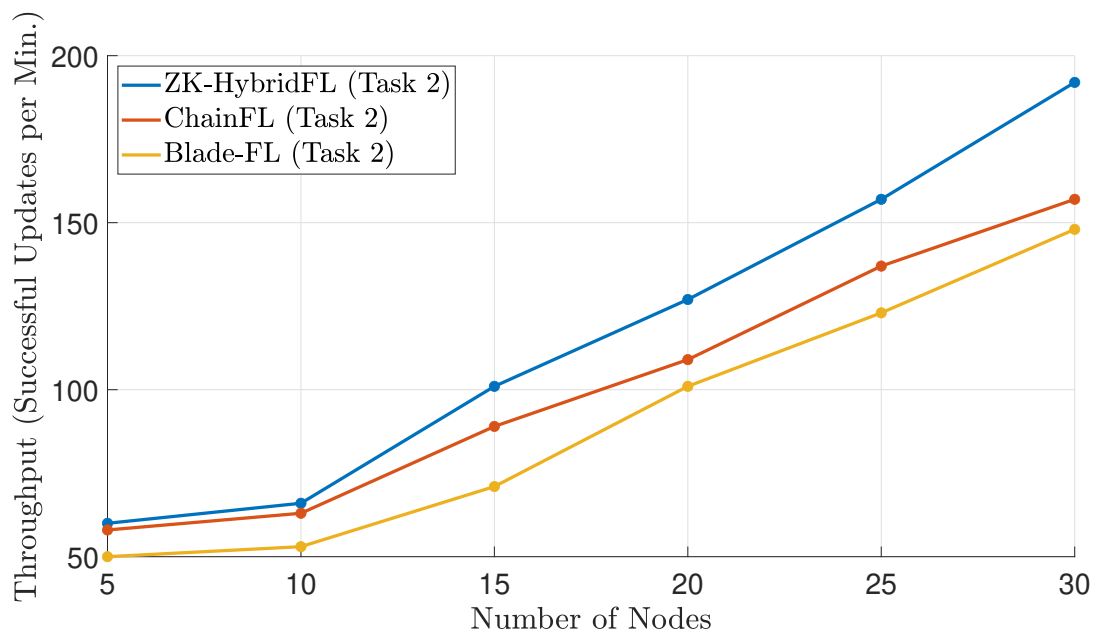


(b) Task 2

Figure 5.14: Latency of Blade-FL, ChainFL, and ZK-HybridFL versus the number of nodes  $n$  with  $\mu = 20\%$  adversarial nodes and  $\gamma = 10\%$  lazy nodes.

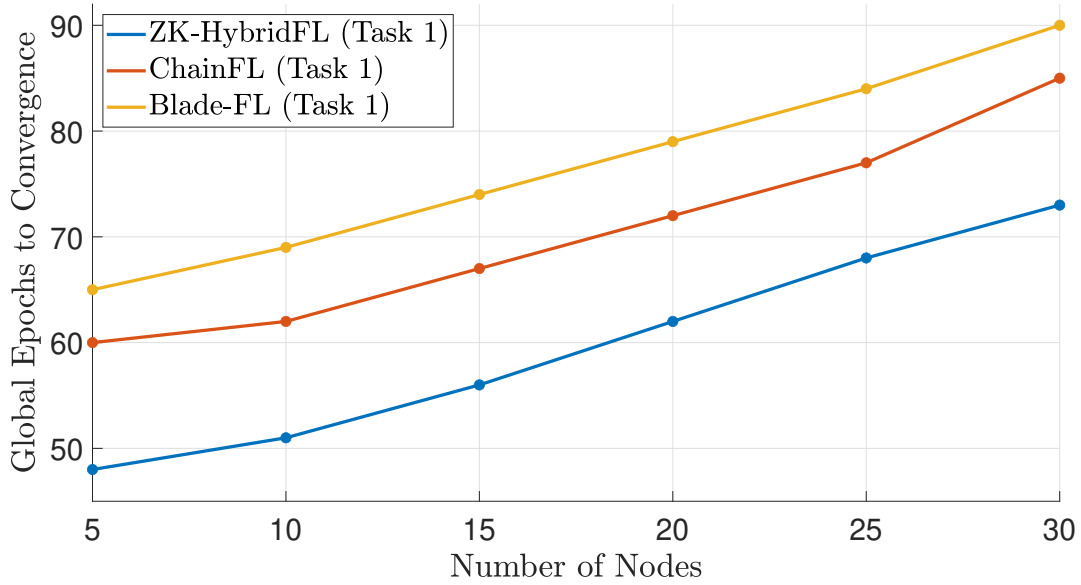


(a) Task 1

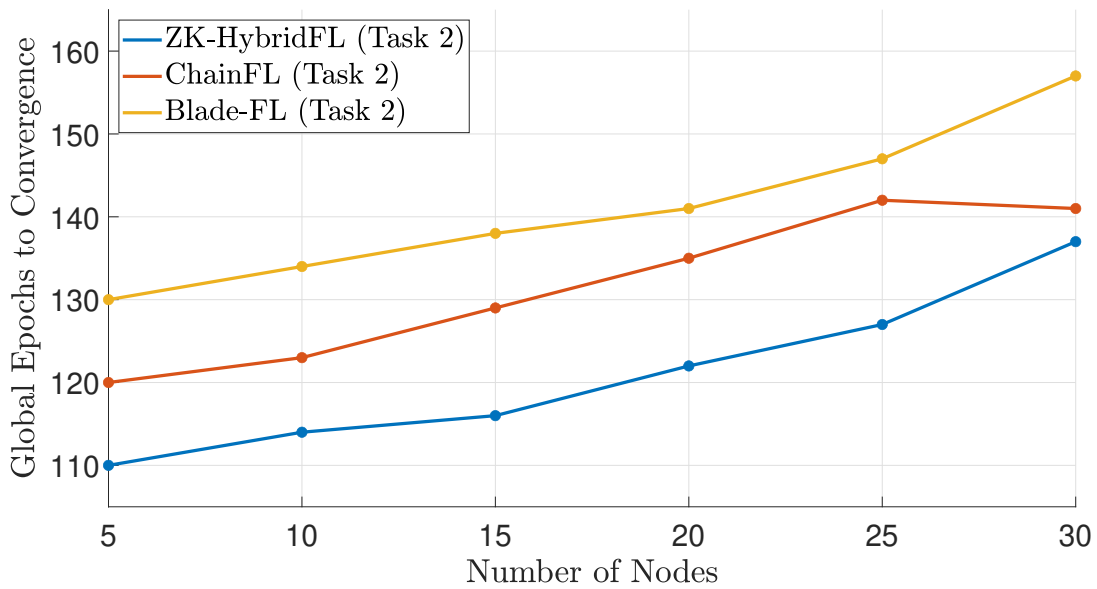


(b) Task 2

Figure 5.15: Throughput of Blade-FL, ChainFL, and ZK-HybridFL versus the number of nodes  $n$  with  $\mu = 20\%$  adversarial nodes and  $\gamma = 10\%$  lazy nodes.



(a) Task 1



(b) Task 2

Figure 5.16: Scalability of Blade-FL, ChainFL, and ZK-HybridFL versus the number of nodes  $n$  with  $\mu = 20\%$  adversarial nodes  $n$  and  $\gamma = 10\%$  lazy nodes.

# Chapter 6: Secure Decentralized Federated Learning via Gossip and Virtual Voting

## 6.1 Introduction

Federated learning (FL) trains a global model across many devices while keeping raw data local and exchanging only model updates, thereby reducing privacy risk and cross-device bandwidth compared with centralized data pooling [230], [231]. In its canonical, hub-and-spoke form, a server orchestrates rounds of federated averaging and may employ secure aggregation so the server observes only encrypted sums [223], [232]. This architecture scales well but concentrates trust and control on the server and can still leak information via gradients if the server is curious or compromised [184], [222].

A natural way to remove the single point of trust is to eliminate the server and let peers exchange updates directly. Classical results on randomized gossip show that repeated neighbor averaging converges at a rate tied to the spectral gap of the communication graph [233]. Building on this foundation, decentralized parallel stochastic gradient descent (D-PSGD) and its asynchronous push-pull variants deliver competitive accuracy and better throughput under heterogeneous bandwidth by avoiding the server bottleneck [234], [235]. Push-sum corrections stabilize training when communication is one-way or the network links change over time [236], and compressed gossip sustains convergence while reducing communication overhead [237]. However, these serverless approaches typically assume honest peers and provide neither global provenance nor a mechanism to *finalize* which updates should influence the model, complicating resilience to Byzantine behavior and stealthy backdoors [208], [238].

Communication structure strongly affects both learning speed and fault containment in decentralized training. Empirical and analytical studies link small-world shortcuts and expander-like

networks to faster mixing, whereas sparse or poorly connected topologies slow agreement and exacerbate stragglers [239]–[241].

A complementary line of work replaces the server with a distributed ledger that records and finalizes updates through consensus. Early concepts and deployments—ranging from on-device FL coordinated by smart contracts to consortium “Swarm Learning”—demonstrate how tamper-evident logs can coordinate rounds and incentivize participation without a trusted server [171], [242]. However, linear blockchains introduce confirmation latency and throughput ceilings that are ill-matched to the cadence of FL [243], [244]. To improve throughput and flexibility, directed acyclic graph (DAG) ledgers broaden the design space; for example, IOTA’s Tangle exploits concurrent approvals to raise throughput [245].

### 6.1.1 Related Works

Gossip-style training underpins decentralized FL without a central server. On the algorithmic side, D-PSGD attains centralized-like convergence while distributing communication across peers [234]; AD-PSGD removes round barriers via asynchronous push–pull with bounded staleness to improve wall-clock efficiency on heterogeneous clusters [235]; and compression-aware gossip (e.g., Choco) reduces bandwidth without sacrificing convergence guarantees [237]. At the system level, GossipFL [246] and FedDual [247] show that neighbor-to-neighbor exchanges can maintain convergence and cut communication, with further applications such as edge workload forecasting [248]. Hybrid semi-decentralized designs like PISCO [249] mix agent-to-agent exchanges with server coordination for efficiency and robustness. Despite their benefits, these approaches typically assume benign peers and provide neither auditability nor per-epoch finality about which updates influence the model.

A separate line of work studies the threats and how to defend against them. Backdoor attacks can implant targeted behavior while keeping overall test accuracy high, and tail/model-replacement attacks show that simple anomaly checks can be evaded [208], [250]. As a result, a poisoned model can pass such checks yet misclassify attacker-chosen inputs. Even when no raw data is shared,

gradient inversion can reconstruct sensitive examples from gradients [184]. To mitigate these risks, Byzantine-robust aggregation rules (e.g., Krum [238], trimmed or coordinate-wise medians [251], and robust federated averaging [252]) limit the impact of bad updates when at most a limited fraction of clients are malicious in a round. Sybil-aware methods down-weight groups of nearly identical updates to curb one attacker using many identities [253]. After training, tools like Neural Cleanse attempt to reverse-engineer potential triggers [254]; this post-hoc detector searches for the smallest mask-and-pattern that induces each target class, with unusually small triggers flagging a likely backdoor.

To make update history auditable and to decide which updates are final, recent systems replace the server with a distributed ledger and consensus. Their validation logic typically evaluates submitted updates on a shared *public validation set* and accepts only those meeting accuracy/loss thresholds before inclusion or reward; the ledger then immutably records both the accepted updates and the aggregation provenance.

Early designs use blockchains and smart contracts to orchestrate aggregation on devices and across institutions, e.g., blockchained on-device FL and Swarm Learning [171], [242], [255], [256]. BLADE-FL integrates a PoW chain where miners verify candidate updates (commonly by testing them on a public validation set) and only then include them in blocks; on-chain transparency can also deter plagiarism and lets one study behavior under lazy clients [244].

To raise throughput and reduce confirmation delay, DAG ledgers allow concurrent approvals and asynchronous aggregation. ChainFL and DAG-EnseFL couple DAG-style ledgering with ensemble workflows, with validation again typically performed on a public set before model admission [257], [258]. DAG-FL proposes a two-layer DAG tailored for on-device FL, adding mechanisms for device asynchrony alongside public-set screening [259]. DAG-BFL studies how learning quality couples with ledger throughput in wireless settings via resource-aware scheduling [260]. Most recently, IronForge’s coordinator-free DAG proposes a novel mechanism for controlling model acceptance via VRF-elected settlement committees and proof-of-learning, while continuing to rely on public validation [170].

### 6.1.2 Motivation and Contributions

Prior work falls into two main camps. First, *serverless gossip* removes the server and achieves efficient neighbor-to-neighbor learning, but typically assumes honest peers and lacks a principled notion of provenance and *finality* for which updates should count in each epoch—complicating resilience to Byzantine behavior, lazy replays, and system-wide auditability. Second, *ledger-assisted FL* introduces a tamper-evident log and incentives; however, linear blockchains reintroduce confirmation latency and global broadcast overhead that are ill-matched to FL’s cadence [243], [244].

DAG ledgers increase concurrency and relax strict serialization, and several FL systems exploit this design. Yet the consensus protocols layered atop DAGs in current FL systems are architecturally incompatible with DFL. The core problem is that these systems treat the DAG as a passive logbook while running consensus separately from the model-exchange history. Nodes exchange model updates and record them on the ledger, but the consensus mechanism ignores this communication structure. To decide which updates are final for aggregation, a separate traditional consensus protocol is invoked, typically requiring every node to maintain a complete global view of the ledger [170], [257], [258]. This requirement fatally undermines DFL’s primary advantage, locality. Gossip-based DFL derives its efficiency from nodes communicating with and processing information from only a small neighborhood. By mandating network-wide awareness and participation in a separate global-state protocol, these schemes bolt a heavyweight global consensus onto a lightweight local one, negating the very efficiency they sought to achieve.

A preferable ledger-based DFL derives consensus directly from the communication graph: the act of gossiping a model is the act of performing consensus. This preserves DFL’s locality—light nodes need only handle their own exchanges—while a system-wide decision emerges from the history of these local interactions. Hashgraph’s gossip-about-gossip with virtual voting shows that asynchronous BFT finality can be achieved without explicit vote messages [9], suggesting an attractive path in which a lightweight gossip-native control plane determines which updates are admitted while the data plane remains neighbor-to-neighbor exchanges. What is missing is a

coherent architecture that realizes this separation without reverting to heavyweight, block-centric global consensus protocols

To fill this gap, we propose *gspDAG-FL*, a gossip-native FL framework in which nodes continue one-shot, neighbor-to-neighbor exchanges each epoch, while a lightweight DAG control plane performs Hashgraph-style virtual voting to decide which updates are accepted for aggregation. Specifically, our primary contributions are as follows:

1. We propose a novel DFL architecture, *gspDAG-FL*, that decouples model exchange from consensus. It uniquely combines a gossip-based data plane for efficient peer-to-peer update dissemination with a Hashgraph-inspired control plane that achieves asynchronous Byzantine fault tolerance. This control plane operates on a compact *Topology DAG* reconstructed by full nodes from lightweight cryptographic proofs, enabling finality through virtual voting without the overhead of explicit vote messages or global block broadcasts.
2. We design and integrate a pluggable multi-layered defense pipeline that provides resilience against a mixed population of lazy and Byzantine adversaries. This pipeline consists of (i) real-time magnitude and directional filtering applied by nodes during gossip to reject malformed updates immediately, and (ii) a novel post-consensus semantic audit where each node uses a small, *private* validation set to detect and discard updates exhibiting anomalous behavior, such as stealthy backdoors, prior to aggregation.
3. We evaluate *gspDAG-FL* on image classification and language modeling. It delivers accuracy for Task 1 and perplexity for Task 2 comparable to leading *ledger-based* FL under mixed adversarial and lazy participation, while improving core system metrics: lower *latency* measured by shorter time per update exchange and consensus round; higher *throughput* measured by more correctly validated honest updates included in each aggregation round; and better *scalability* reflected in fewer consensus rounds to reach target accuracy or perplexity as the network grows. These gains show that our gossip-native, DAG-driven gossip control plane removes confirmation bottlenecks in ledger-based FL schemes and converts added time

and participants into faster, more efficient learning.

The remainder of the chapter is organized as follows. Sec. 6.2 reviews FL protocols, covering centralized, decentralized gossip-based, and ledger-backed variants. Sec. 6.3 presents gspDAG-FL our DFL framework over a Hashgraph DAG, detailing the system model, node roles, and overall workflow. Sec. 6.4 specifies the control plane and consensus mechanism in gspDAG-FL, including virtual voting, finality, and validations prior to aggregation. Sec. 6.5 describes the experimental setup and some baseline schemes, and reports results on learning and ledger metrics. Sec. 6.6 concludes and outlines future work.

## 6.2 Federated Learning Protocols

### 6.2.1 Conventional FL

Suppose that  $N$  nodes participate in FL. Each node  $i$  holds a local dataset  $\mathcal{D}_i$  and defines a local loss function  $\mathcal{L}_i(x; \theta)$  where  $x \in \mathcal{D}_i$  is a data sample and  $\theta$  represents the model parameter. The goal is to find the optimal global parameter  $\theta$  that minimizes the average local loss:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^N \mathbb{E}_{x \sim \mathcal{D}_i} [\mathcal{L}_i(x; \theta)]. \quad (6.1)$$

In conventional FL (e.g., [223], [230]), as shown in Fig.6.1(a) there is a central server and the FL is enabled by two-way communication between the central server and the  $N$  nodes. In each epoch  $t$ , every node  $i$  starts from the current global model  $\theta_i^{t,0} = \theta^{t-1}$  and performs  $K$  steps of local stochastic gradient descent (SGD):

$$\theta_i^{t,k+1} = \theta_i^{t,k} - \eta \nabla \mathcal{L}_i(x_i^{t,k}; \theta_i^{t,k}), \quad k = 0, 1, \dots, K-1, \quad (6.2)$$

where  $\eta > 0$  is the local learning rate and  $x_i^{t,k}$  denotes a random mini-batch sampled from  $\mathcal{D}_i$ .

After  $K$  iterations, each node  $i$  sends its local model  $\theta_i^{t,K}$  to the central server, which aggregates

them to form the global model

$$\theta^t = \sum_{i=1}^N w_i \theta_i^{t,K}, \quad \text{with } w_i > 0 \quad \text{and} \quad \sum_{i=1}^N w_i = 1, \quad (6.3)$$

that is then broadcast to all nodes. This iterative process continues until convergence.

### 6.2.2 Gossip-based Decentralized FL

Conventional FL, though effective, relies on a single server, making it vulnerable to tampering, inversion, and denial-of-service attacks. Gossip-based DFL removes this central hub by enabling peer-to-peer exchanges in which each node randomly shares and aggregates updates with a small subset of neighbors. By distributing both aggregation and coordination, it eliminates the single point of failure, shrinks the attack surface, and strengthens resilience against adversarial or malfunctioning nodes.

In traditional gossip protocols [233], [236], nodes repeatedly execute push–pull exchanges of some messages from a fixed set with their neighbors, aggregating at each iteration until every participant converges to an identical set of messages. A naive approach to DFL is to implement the gossip for each epoch at full scale, such that progress to the next epoch occurs only when full diffusion has been achieved. A more efficient approach is to invoke a one-shot gossip per epoch: after local updates, each node transmits its model parameters exactly once to its immediate neighbors, simultaneously receives theirs, and computes a single weighted average. While this solitary exchange does not enforce instantaneous consensus, iterating it over successive federated epochs drives all local models toward agreement under standard connectivity conditions, thereby obviating the need for an intra-epoch convergence loop [234], [235], [237].

Specifically, as shown in Fig. 6.1(b), in fully DFL, each node  $i$  communicates only with its neighbor nodes, denoted by  $\mathcal{N}_i$ . At epoch  $t$ , each node  $i$  executes the following steps:

**Step 1: Local Model Update:** Each node  $i$  initializes with the aggregated model from the

previous epoch  $\theta_i^{t,0} = \theta_i^{t-1}$  and performs  $K$  steps of local SGD:

$$\begin{aligned} \theta_i^{t,k+1} &= \theta_i^{t,k} - \eta \left( \nabla \mathcal{L}_i(x_i^{t,k}; \theta_i^{t,k}) - \hat{g}_i^{t-1} + \frac{1}{\lambda} (\theta_i^{t,k} - \theta_i^{t-1}) \right), \\ k &= 0, \dots, K-1, \end{aligned} \quad (6.4)$$

where  $\eta > 0$  is the local learning rate,  $x_i^{t,k}$  is a mini-batch sampled from  $\mathcal{D}_i$ ,  $\hat{g}_i^{t-1}$  is a dual variable from the previous epoch, and  $\lambda > 0$  is a penalty parameter. The term,  $-\hat{g}_i^{t-1} + \frac{1}{\lambda} (\theta_i^{t,k} - \theta_i^{t-1})$  in (6.4) stems from the augmented-Lagrangian's consensus penalty:  $-\hat{g}_i^{t-1}$  is the dual variable enforcing agreement with the previous global anchor  $\theta_i^{t-1}$ , while  $\frac{1}{\lambda} (\theta_i^{t,k} - \theta_i^{t-1})$  is the gradient of the quadratic penalty  $\frac{1}{2\lambda} \|\theta - \theta_i^{t-1}\|^2$ , which acts as a proximal pull to limit drift away from the last synchronized model. Together, these corrections inject global-consensus information and anchor each inner-loop SGD update so that  $\theta_i^{t,k}$  remains close to  $\theta_i^{t-1}$ .

At the end of the iteration, node  $i$  obtains its updated model  $\theta_i^{t,K}$ . To promote consistency and mitigate drift, it then updates its dual variable and computes an intermediate update:

$$\hat{g}_i^t = \hat{g}_i^{t-1} - \frac{1}{\lambda} (\theta_i^{t,K} - \theta_i^{t-1}), \quad (6.5)$$

$$z_i^t = \theta_i^{t,K} - \lambda \hat{g}_i^{t-1}. \quad (6.6)$$

The intermediate update  $z_i^t$  captures both the new model parameter and a corrective adjustment based on prior discrepancies, and is the message to be sent to the neighbor nodes in  $\mathcal{N}_i$ .

**Step 2: Local Message Exchange:** Each node  $i$  simultaneously sends  $z_i^t$  to all its neighbor nodes in  $\mathcal{N}_i$ . Then each node  $i$  receives the intermediate updates  $z_j^t$  from all its neighbors  $j \in \mathcal{N}_i$ .

**Step 3: Local Model Aggregations:** Each node  $i$  forms its aggregated local model  $\theta_i^t$  based on

the messages received from its neighbors as:

$$\theta_i^t = \sum_{j \in \mathcal{N}_i} w_{ij} z_j^t, \quad \text{with } w_{ij} > 0 \quad \text{and} \quad \sum_{j \in \mathcal{N}_i} w_{ij} = 1. \quad (6.7)$$

### 6.2.3 FL over Blockchain

Blockchain integration provides a tamper-proof, consensus-driven ledger for recording the model updates and built-in incentive mechanism for validating and rewarding nodes [243], [244]. In this case, the underlying communication model is in the form of all-to-all broadcast, as shown in Fig.6.1(c). Specifically, in each epoch, after training on mini-batches from (6.2), node  $i$  signs and broadcasts its updated model  $\theta_i^{t,K}$  as a blockchain transaction to all other nodes. Every node then verifies its received updated local models against a public dataset using a loss threshold and aggregates the validated updates to form a global model. As a result, multiple versions of the global model may emerge depending on the selected updated local models by different nodes. Nodes subsequently compete to mine a new block using Proof of Work (PoW), where the block records both the verified transactions and the aggregated model, ultimately determining which node's global model will be adopted for the next epoch. Once the block is validated network-wide, the winning global model is adopted as  $\theta^t$ , and the cycle repeats until convergence.

While conventional FL requires a trusted central server and existing blockchain-based solutions merely transplant the hub-and-spoke aggregation of conventional FL onto a distributed ledger, gossip-based FL is inherently fully decentralized, where each node exchanges and averages updates only with a select peer set. Although several works have explored conventional FL over blockchain, to date there is no system that integrates the gossip-based DFL on-chain. By fusing true gossip-based model updates with a distributed ledger backend for immutable recording and consensus, our approach fills this critical gap and lays the groundwork for a resilient, secure, and scalable federated learning framework.

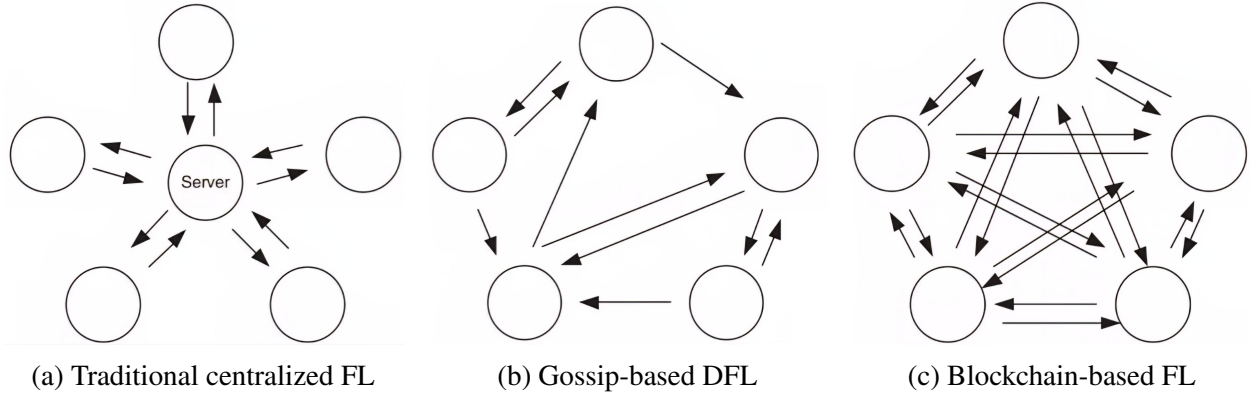


Figure 6.1: Communication protocols for different FL approaches.

### 6.3 Gossip-based FL Over Hashgraph DAG (gspDAG-FL)

#### 6.3.1 System Architecture

In Sec. 6.2.2 we showed how gossip-based DFL eliminates a central server by relying solely on peer-to-peer exchanges. Motivated by the prior work that runs conventional FL on blockchains, we ask whether one can likewise run fully DFL over a distributed ledger—a fusion that, to our knowledge, has not been proposed. A naïve integration of federated gossip with existing blockchain or DAG platforms (e.g., standard PoW chains or IOTA’s Tangle) reintroduces all-to-all broadcasts, leading to excessive overhead, and negating gossip’s efficiency.

To bridge these ideas, we now turn our attention to how the Hashgraph DAG structure [9] can be leveraged to support gossip based DFL.

#### Hashgraph DAG

Each learning epoch  $t$  is divided into  $R_t$  ripples—short micro-cycles in which data-plane model exchanges and control-plane messages proceed in parallel, ensuring uninterrupted dissemination. In ripple  $r \in \{1, \dots, R_t\}$ , for each node  $i$ , if it receives a model in ripple  $r - 1$ , then it gossips the received model to a randomly chosen neighbor in  $\mathcal{N}_i$ . The epoch ends and advances to  $t + 1$  once, for each node  $i$ , a sufficient number of models are received and confirmed via consensus by all full nodes, enabling node  $i$  to compute the aggregated model  $\theta_i^t$ .

In Hashgraph DAG, vertices represent events created by nodes. For epoch  $t$ , each node  $i$  creates event  $e_i^t(r)$  in ripple  $r$ . The genesis event  $e_i^t(0)$  contains the model  $z_i^t$  trained by node  $i$  in that epoch. For  $r \geq 1$ , event  $e_i^t(r)$  may contain a model received from a neighbor in  $\mathcal{N}_i$  or be empty. Specifically, if node  $i$  receives model  $z_\kappa^t$  originally trained by node  $\kappa$  in epoch  $t$ , from a node  $j \in \mathcal{N}_i$  during ripple  $r$ , then  $e_i^t(r)$  contains  $z_\kappa^t$ ; and if no node in  $\mathcal{N}_i$  gossiped to node  $i$  in ripple  $r$ , then  $e_i^t$  is empty. Note that  $\kappa = j$  for first-hand gossip. If node  $j$  forwards a model it received that is originally trained by node  $j' \in \mathcal{N}_j$ , then  $\kappa = j'$ ; and if the model was originally trained by node  $j'' \in \mathcal{N}_{j'}$  and forwarded by node  $j' \in \mathcal{N}_j$ , and  $j \in \mathcal{N}_i$ , then  $\kappa = j''$ , and so on through any number of hops.

In a Hashgraph DAG, nodes represent events and are on a rectangle grid, where the  $i$ -th column represents events created by node  $i$ , and the  $r$ -th row corresponds to events created at ripple  $r$ . The edges are between events in consecutive ripples. In particular, each non-empty event  $e_i^t(r)$  for  $r \geq 1$  has two edges: a dashed edge linking to its own predecessor  $e_i^t(r-1)$  in the same column  $i$ , and a solid edge linking to its parent event  $\mathcal{P}\{e_i^t(r)\}$ , defined as the event  $e_j^t(r-1)$  in column  $j$ , where  $j \in \mathcal{N}_i$  is the neighbor that gossiped to node  $i$  in ripple  $r$ . Here, be noted that an event  $e_i^t(r)$  and its parent  $\mathcal{P}\{e_i^t(r)\}$  carry the same model. Also, if event  $e_i^t(r)$  is empty, then it has no parent.

Fig. 6.2 illustrates a Hashgraph DAG with  $N = 4$ , nodes with the underlying connectivity among the four nodes also shown. For instance, event  $e_1(1)$  links to its parent  $\mathcal{P}\{e_1(1)\} = e_3(0)$ , which means that in ripple  $r = 1$ , node 3, having trained model  $z_3^t$ , gossiped its update  $z_3^t$ , the payload of  $e_3(0)$ , to node 1 and thereby created event  $e_1(1)$ . Node 1 then validates  $z_3^t$  and, upon finding it valid, forwards it to node 2 in ripple 2, producing event  $e_2(2)$ . Thus, the edge from  $e_3(0)$  to  $e_1(1)$  represents a first-hand gossip, whereas the edge from  $e_1(1)$  to  $e_2(2)$  is a gossip-of-gossip.

Also, node 2's model  $z_2^t$  fails validation by node 4 in ripple 1. Consequently, node 4 rejects  $z_2^t$  due to failed validity and does not transmit it to its neighbor in ripple 2; hence, there is no outgoing edge from  $e_4(1)$  to any of the events in ripple 2. Moreover, because node 3 receives no gossip in ripple 2, it generates an empty event  $e_3(2)$ .

In gspDAG-FL, we distinguish between two node types: light nodes and full nodes. All nodes perform the light-node duties described in Sec. II-B, while full nodes additionally handle the

responsibilities outlined in Sec. II-C. To support these roles, all nodes communicate over a data channel to exchange data with neighbors, while a control channel facilitates consensus-related coordination among full nodes and between full and light nodes.

### 6.3.2 Light-node Duties

#### Storage

Each node  $i$  maintains only the portion of the Hashgraph DAG corresponding to its own sequence of events in column  $i$ , along with the parent events of those it creates. Specifically, this subgraph includes all events  $e_i^t$  created by node  $i$ , as well as their respective parents  $\mathcal{P}(e_i^t) = e_j^t$ , where each parent  $e_j^t$  is an event created by a neighbor  $j \in \mathcal{N}_i$ . Each stored event carries its full payload including the model update  $z_i^t$ , signature, and metadata. Node  $i$  uses this information for local training, validation, and gossiping.

#### Communication

At each ripple  $r$ , every node communicates over two distinct channels: the data channel, where full event payloads are exchanged among immediate neighbors, and the control channel, which facilitates communication between full and light nodes for exchanging lightweight topology information. The control channel is also used by full nodes for consensus coordination.

On the data channel, at the start of ripple  $r$ , if node  $i$  holds a model and a corresponding proof of gossip received and locally validated during ripple  $r - 1$ . It selects a neighbor  $j \in \mathcal{N}_i$ , generates a fresh proof  $\pi_i(r)$  attesting to its own gossip action, and sends both the model and  $\pi_i(r)$  to node  $j$ .

Separately, node  $i$  also receives a model and a proof  $\pi_{j'}(r)$  from a neighbor  $j' \in \mathcal{N}_i$ . It verifies  $\pi_{j'}(r)$  and validates the model. If both checks succeed, it creates a new event  $e_i(r)$  embedding the received model. If either check fails,  $e_i(r)$  is created as an empty event.

On the control channel, node  $i$  sends the validated proof  $\pi_{j'}(r)$  to all full nodes as evidence that  $e_{j'}^t(r - 1)$  is indeed the parent of its own event  $e_i^t(r)$ , i.e.,  $\mathcal{P}\{e_i^t(r)\} = e_{j'}^t(r - 1)$ . This allows full nodes to verify the correctness of the parent relationship, and augment the Topology DAG.

### 6.3.3 Full-node Duties

#### **Storage**

In Hashgraph, confirmation of events emerges from virtual voting on the *Topology DAG*, the graph that records each node's choice of gossip neighbors. When node  $i$  selects neighbor  $j$  to gossip to, the newly created event records node  $j$  as its parent, thereby augmenting the global parent-link graph and impacting which events achieve confirmation. Light nodes retain only the portion of the Hashgraph DAG corresponding to nodes containing models that they create or receive so they lack a complete Topology DAG from which to derive confirmations on their own.

Full nodes, by contrast, do not unilaterally dictate consensus; rather, they build the complete Topology DAG by gathering and merging the structural slices offered by light nodes. In doing so, full nodes leverage collective information from across the network to assemble the global interaction graph. This collaborative reconstruction is made possible by an all-to-all broadcast among full nodes and preserves true decentralization: no single node ever holds all event payloads, yet every full node is equipped with the complete link structure required to perform virtual voting and finalize consensus in a Byzantine-fault-tolerant fashion.

#### **Communication and Consensus Duties**

Both light and full nodes perform the same gossip on the data channel: each generates one event per epoch, validates incoming events, and forwards only those that pass. Full nodes then switch to the control channel, where each full node listens for every gossip-proof  $\pi$  and parent link, validates it, and augments its own Topology DAG. After integrating all gossip events for ripple  $r$ , each full node *locally* executes virtual voting to produce a provisional “yes/no” vector over the  $N$  models and broadcasts this vector to all other full nodes over the control channel. The decision on whether the current epoch terminates, and if so, which models become finally confirmed is made *collectively* by all full nodes.

### 6.3.4 Workflow of DFL over Hashgraph DAG

In our proposed gspDAG-FL over Hashgraph, each epoch  $t$  is divided into three stages. Stages 1 and 3 are executed once per epoch, while Stage 2 comprises three sub-steps that are iteratively executed over the  $R_t$  ripples.

#### **Stage 1: Local model update**

As in Sec. I-B, at the start of each epoch  $t$ , every node  $i$  initializes its model as  $\theta_i^{t,0} = \theta_i^{t-1}$  and performs local updates via (6.4)–(6.6), yielding the model  $z_i^t$  to be used in subsequent gossip exchanges.

#### **Stage 2: Gossip over ripples (data exchange, topology sync, and consensus)**

##### **Step 1: Data–Channel Gossip:**

At the start of ripple  $r \geq 1$ , for node  $i$ , if  $e_i^t(r-1)$  is not empty, then it sends the model contained in  $e_i^t(r-1)$  with a fresh proof  $\pi_i(r)$  to a random neighbor over the data channel; otherwise it sends nothing. Then node  $i$  listens for at most one incoming gossip. If it receives a model with a proof  $\pi_{j'}(r)$ , it verifies the proof (see Sec. 6.4.2) and validates the model (see Sec. 6.4.21); on success, it logs  $e_i^t(r)$  embedding the received model, otherwise it logs an empty event. Each event  $e_i^t(r)$  carries: (i) a model—either  $z_i^t$  if  $r = 1$ , or the validated update from ripple  $r - 1$ ; (ii) the gossip-proof  $\pi_{j'}(r)$ ; and (iii) metadata: creator ID  $i$ , epoch  $t$ , ripple  $r$ .

##### **Step 2: Control–Channel Topology Sync:**

After the data-channel phase of ripple  $r$  ends, nodes switch to the control channel. Any node  $i$  that receives a gossip-proof  $\pi_{j'}(r)$  first validates it and, upon validity, rebroadcasts  $\pi_{j'}(r)$  to all full nodes. If  $\pi_{j'}(r)$  is invalid then node  $i$  sends nothing to full nodes. Each full node listens continuously, validates each  $\pi_{j'}(r)$  by checking its digital signature and parent link (see Sec. 6.4.22), and augments its Topology DAG, discarding invalid proofs and penalizing their senders. Once all

valid  $\pi_{j'}(r)$  for ripple  $r$  have been integrated, the Topology DAG is complete and the full node is ready to perform virtual voting.

### Step 3: Gossip Termination Check by Full Nodes

All full nodes monitor their respective Topology DAGs and run virtual voting to decide both when to end the gossip phase and which models are confirmed for aggregation. When they reach consensus on a set of confirmed models, using control channel they inform all nodes to proceed to Stage 3 for model aggregation (see Sec. IV.A for details).

#### Stage 3: Semantic audit and model aggregation

Once the gossip loop ends, all nodes enter the last stage. Recall that each node  $i$  maintains only its own column of the Hashgraph DAG plus the parent events it has stored. Each node  $i$  forms a set  $\mathcal{S}_i^t$  consisting of updates  $z_j^t$  that both appear in the list of confirmed models by the full nodes and reside in node  $i$ 's local subgraph. Node  $i$  then applies the semantic-consistency audit for removing the stealthy models (see Sec. 6.4.2), yielding  $\tilde{\mathcal{S}}_i^t \subseteq \mathcal{S}_i^t$ . Finally, it performs model aggregation via weighted averaging:

$$\theta_i^t = \sum_{j \in \tilde{\mathcal{S}}_i^t} \tilde{w}_{ij}^t z_j^t, \quad \tilde{w}_{ij}^t = \frac{w_{ij}}{\sum_{k \in \tilde{\mathcal{S}}_i^t} w_{ik}}. \quad (6.8)$$

## 6.4 Model Validation and Consensus

### 6.4.1 Consensus Process by Full Nodes

In gspDAG-FL, only the events created in the genesis ripple, i.e., model  $z_i^t$  embedded in the event  $e_i^t(r=0)$ ,  $i = 1, \dots, N$ , are subject to confirmation by full nodes. Similar to other distributed ledger systems, each node, whether a full node or a light node, participates in voting to confirm these models. However, our approach is based on virtual voting, where votes are inferred rather than explicitly cast. This subsection explains the virtual voting process. In particular, we describe how

each full node determines locally which models are confirmed, and how all full nodes collectively decide when to terminate gossiping and finalize the set of confirmed models for aggregation.

In virtual voting, a node does not directly cast a vote. Instead, for an event  $e_j^t(r)$  created by node  $j$ , the structure of the Topology DAG, i.e., the parent relationship, determines whether node  $j$  in ripple  $r$  confirms the validity of the model  $z_i^t$  in  $e_i^t(r=0)$ . The confirmation decision is inferred from the position of the event  $e_i^t(r)$  in the Topology DAG of a full node. In what follows, we first introduce a number of quantities used in the consensus process, and then describe the key steps of the consensus process by all full nodes to confirm the local model  $z_i^t$  at epoch  $t$ . For notational simplicity, we drop the epoch index  $t$  in the rest of this section.

## Definitions

At the beginning of ripple  $r \geq 1$  of Step 3 of Stage 2, each full node has an updated Topology DAG. Then for each most recent (non-empty) event  $e_j(r)$ ,  $j = 1, \dots, N$ , the full node updates a number of quantities, defined as follows.

First, we introduce the moment of event  $e_j(r)$ , denoted by  $\mathcal{M}\{e_j(r)\}$ , and the earliest reachable set  $\mathcal{R}\{e_j(r)\}$ , which are jointly updated recursively over ripple  $r$ . At the genesis ripple  $r = 0$ , we set  $\mathcal{M}\{e_j(0)\} = 1$  and  $\mathcal{R}\{e_j(r)\} = \emptyset$ , for  $j = 1, \dots, N$ . For  $r \geq 1$ , if  $e_j(r)$  is empty, then  $\mathcal{M}\{e_j(r)\} = 0$ . Next, for  $r \geq 1$  and a non-empty event  $e_j(r)$ ,  $\mathcal{R}\{e_j(r)\}$  is defined as the set of events, each created by a distinct node and with moment value

$$M = \max (\mathcal{M}\{e_j(r-1)\}, \mathcal{M}\{\mathcal{P}\{e_j(r)\}\}), \quad (6.9)$$

that are reachable from  $e_j(r)$  along dashed or solid edges in the Topology DAG. If multiple such events exist from a same node, only the one with the smallest ripple index  $r$  is included. In this way,  $\mathcal{R}\{e_j(r)\}$  represents the set of earliest reachable events of moment  $M$ , one per node, from the perspective of  $e_j(r)$ .

Moreover, for  $r \geq 1$ , using  $\mathcal{R}\{e_j(r)\}$ , we classify a non-empty event  $e_j(r)$  as follows:

$$e_j(r) : \begin{cases} \text{voting event,} & \text{if } |\mathcal{R}\{e_j(r)\}| \geq \frac{2N}{3}, \\ \text{non-voting event,} & \text{otherwise.} \end{cases} \quad (6.10)$$

Then the moment of  $e_j(r)$  is set as

$$\mathcal{M}\{e_j(r)\} = \begin{cases} M + 1, & \text{if } e_j(r) \text{ is voting,} \\ M, & \text{otherwise.} \end{cases} \quad (6.11)$$

For example, in Fig. 6.2 consider event  $e_1(4)$ . Its parent is  $\mathcal{P}\{e_1(4)\} = e_2(3)$ . The moments of all nodes can be calculated according to (6.9)-(6.11) for  $r = 1, 2, 3$  and are color coded as in Fig. 6.2. To decide whether  $e_1(4)$  is a voting event, we first calculate its moment using (6.9), as  $M = \max(\mathcal{M}\{e_1(3)\}, \mathcal{M}\{e_2(3)\}) = 2$ . Then we form the set of earliest reachable events with moment 2:

$$\mathcal{R}\{e_1(4)\} = \{e_1(2), e_2(2), e_4(2)\}.$$

Since  $|\mathcal{R}\{e_1(4)\}| = 3$  exceeds the supermajority threshold  $2N/3 = 8/3$  for  $N = 4$ , by (6.10) we classify  $e_2(4)$  as a voting event and using (6.11) we set its moment value to

$$\mathcal{M}\{e_2(4)\} = 2 + 1 = 3.$$

Note that although  $e_3(3)$  is also a moment-2 voting event, it is not in  $\mathcal{R}\{e_1(4)\}$  because there is no path from  $e_1(4)$  to  $e_3(3)$ .

For a voting event  $e_j(r)$ , its virtual vote on the validity of each model  $z_i$  is recursively defined as

follows: for  $i = 1, \dots, N$ ,

$$\mathcal{V}(e_j(r), z_i) = \begin{cases} 1, & \text{if } \mathcal{M}\{e_j(r)\} = 2 \quad \text{and} \quad z_i \in \mathcal{R}\{e_j(r)\}, \\ 1, & \text{if } \mathcal{M}\{e_j(r)\} > 2 \quad \text{and} \quad \sum_{e \in \mathcal{R}\{e_j(r)\}} \mathcal{V}(e, z_i) > \frac{|\mathcal{R}\{e_j(r)\}|}{2}, \\ 0, & \text{otherwise.} \end{cases} \quad (6.12)$$

Recall that for a voting event  $e_j(r)$ , if  $\mathcal{M}\{e_j(r)\} = M$  then for each  $e \in \mathcal{R}\{e_j(r)\}$ ,  $\mathcal{M}\{e\} = M - 1$ .

Therefore, the above virtual vote is recursively defined over the moment value.

Furthermore, associated with a voting event  $e_j(r)$  there is an intermediate vote on each model  $z_i, i = 1, \dots, N$ , computed based on the virtual votes by events in the set  $\mathcal{R}\{e_j(r)\}$ , given by:

$$\mathcal{U}(e_j(r), z_i) = \begin{cases} 1, & \text{if } \mathcal{M}\{e_j(r)\} > 2 \quad \text{and} \quad \sum_{e \in \mathcal{R}\{e_j(r)\}} \mathcal{V}(e, z_i) \geq \frac{2N}{3}, \\ 0, & \text{otherwise.} \end{cases} \quad (6.13)$$

Finally, let  $S_j(r)$  denote the set of models that node  $j$  has seen through gossip up to ripple  $r$ . At  $r = 0$ , it contains only the node's own local model, i.e.,  $S_j(0) = \{z_j\}$ . As  $r$  increases,  $S_j(r)$  grows whenever node  $j$  receives a new model via gossip, i.e.,  $S_j(r) = S_j(r - 1) \cup \{z_k\}$ , where  $z_k$  is the model in  $e_j(r)$ .

### Consensus: full-node local computations

Let  $C(r)$  denote the set of finally confirmed models by the end of ripple  $r$ , with  $C(0) = \emptyset$ . Then at the beginning of ripple  $r$ , the set of unconfirmed models is the complement set of  $C(r - 1)$ , i.e.,  $\bar{C}(r - 1)$ . All below computations in ripple  $r$  are performed only for  $z_i \in \bar{C}(r - 1)$ ; confirmed models in  $C(r - 1)$  are excluded and never reconsidered in later ripples.

In ripple  $r \geq 1$ , each full node begins by computing the earliest reachable set  $\mathcal{R}\{e_j(r)\}$  of each

non-empty event  $e_j(r)$  created in ripple  $r$ , classifying it as either voting or non-voting according to (6.10), and set its moment according to (6.11). Then, for each model  $z_i \in \bar{C}(r-1)$ , the full node computes the corresponding virtual votes  $\mathcal{V}(e_j(r), z_i)$  and intermediate votes  $\mathcal{U}(e_j(r), z_i)$  for each voting event  $e_j(r)$ , using (6.12) and (6.13), respectively. In addition, for each node  $j = 1, \dots, N$ , the full node also updates the model set  $S_j(r)$  based on the model in  $e_j(r)$ .

Next, the full node determines whether each unconfirmed model  $z_i \in \bar{C}(r-1)$  is *locally confirmed* from its own perspective. Specifically, if there exists at least one voting event  $e_j(r)$  such that  $\mathcal{U}(e_j(r), z_i) = 1$ , then the full node considers  $z_i$  to be locally confirmed. Otherwise, it marks  $z_i$  as unconfirmed.

As a result, the full node obtains a binary vector of length  $|\bar{C}(r-1)|$ , indicating the local confirmation status of each model in  $\bar{C}(r-1)$ .

### **Consensus: full-node exchanges over control channel**

Every full node broadcasts its local confirmation status vector to all other full nodes. If at least a fraction  $\beta$  of the full nodes have locally confirmed model  $z_i \in \bar{C}(r-1)$ , then it is *finally confirmed*. As a result, all full nodes augment  $C(r-1)$  with the newly confirmed models in  $\bar{C}(r-1)$ , to obtain  $C(r)$ .

A full node considers node  $j$  to be *ready* if at least  $Q$  models in its local DAG are finally confirmed, i.e.,

$$|C(r) \cap S_j(r)| \geq Q. \quad (6.14)$$

If a full node marks at least  $\beta$  of all nodes as ready, then it broadcasts  $C(r)$  with its own signature to all nodes over the control channel. If a node  $j$  receives the common  $C(r)$  from at least a fraction  $\beta$  of the full nodes, then it stops the gossiping stage and computes its final confirmed model set  $S_j^t = C(r) \cap S_j(r)$  and proceeds to the model aggregation stage. Otherwise, the gossip continues in ripple  $r+1$ . Note that under assumption that two-thirds of the full nodes are honest, the above protocol guarantees that in each ripple, all nodes reach the same confirmation decision.

## 6.4.2 Validations

In Sec. 6.4.2 we describe the payload validation procedure, which rejects malformed or tampered models before gossip. In Sec. 6.4.2, we discuss proof validation where control-channel messages are checked for correct signatures and parent link consistency before being merged into the Topology DAG. Finally, Sec. 6.4.2 presents the post-ripple semantic consistency audit, which evaluates confirmed models on a private validation set to block stealthy behavioral anomalies prior to aggregation.

### Payload validation

Recall that at ripple  $r$  of epoch  $t$ , if node  $i$  receives a model update  $z_\kappa$  from a neighbor node, it validates it before forwarding it to another neighbor at ripple  $r + 1$ . We adopt the validation method in [261], [262].

The basic idea is to first define a metric  $f(z)$  for a model  $z$ . Then the mean  $\mu_i$  and standard deviation  $\sigma_i$  of  $f(z)$  over the models in the final validated model set  $\tilde{\mathcal{S}}_i^{t-1}$  in epoch  $t - 1$  are computed, based on which some threshold parameters are calculated. Finally, to determine the validity of  $z_\kappa$ ,  $f(z_\kappa)$  is compared with the thresholds.

In particular, two metrics are used. The first metric is

$$f_1(z) = \|z - \theta_i^{t-1}\|_2,$$

where  $\theta_i^t$  is given in (6.8).  $f_1(z)$  measures how much the new update  $z_\kappa$  differs in size from the previous consensus model in  $\tilde{\mathcal{S}}_i^{t-1}$ . A  $f_1(z_\kappa)$  close to zero, i.e.,  $f_1(z_\kappa) < L_i^1$  where  $L_i^1$  is a lower threshold determined by  $(\mu_i^1, \sigma_i^1)$ , indicates negligible or repeated updates that add no new information and may come from a lazy node. Very large  $f_1(z_\kappa)$ , i.e.,  $f_1(z_\kappa) > U_i^1$  where  $U_i^1$  is an upper threshold determined by  $(\mu_i^1, \sigma_i^1)$ , reveals unusually large perturbations that could result from noisy or malicious attacks.

The second metric is

$$f_2(z) = \frac{\langle z - \theta_i^{t-1}, \bar{d}_i^{t-1} \rangle}{\|z - \theta_i^{t-1}\|_2},$$

where

$$d_i^{t-1} = \frac{1}{|\tilde{\mathcal{S}}_i^{t-1}|} \sum_{z \in \tilde{\mathcal{S}}_i^{t-1}} \frac{z - \theta_i^{t-1}}{\|z - \theta_i^{t-1}\|_2}, \quad \bar{d}_i^{t-1} = \frac{d_i^{t-1}}{\|d_i^{t-1}\|_2}.$$

$f_2(z)$  measures the *directional alignment* of the incoming update with the consensus direction formed by previous validated updates. Strong alignment indicates the update reinforces the collective direction, while weak alignment suggests it deviates and may be anomalous. Hence,  $z_\kappa$  is rejected if  $f_2(z_\kappa) < L_i^2$ .

### Proof validation

Recall from Sec. II-B that if at ripple  $r$ , node  $i$  receives a model update and a proof  $\pi_{j'}(r)$  from neighbor  $j' \in \mathcal{N}_i$ . It verifies  $\pi_{j'}(r)$  and validates the model; if both checks succeed, it embeds the update in a new event  $e_i(r)$  and on the control channel sends  $\pi_{j'}(r)$  to all full nodes. When a full node receives the proof  $\pi_{j'}(r)$  corresponding to the new event  $e_i(r)$ , it verifies  $\pi_{j'}(r)$ . Both proof verifications are performed by public-key cryptography adapted from the original Hashgraph protocol: each node in the network has a *secret key* known only to itself and a corresponding *public key* shared with others. The proof  $\pi_{j'}(r)$  is essentially a digital signature generated using node  $j'$ 's secret key and includes a message indicating that node  $j'$  is the sender of the gossip to node  $i$  in ripple  $r$  of epoch  $t$ .

The verifying node uses node  $j'$ 's public key to *verify* the signature. If the verification succeeds, that is, the signature is mathematically valid for the message and the public key, the proof is accepted and  $e_{j'}(r-1)$  is recorded as the parent of  $e_i(r)$  in each full node's Topology DAG. Otherwise,  $e_j(r)$  is an empty event. For implementation details and security analyses of these signature schemes and consensus protocols, see [9], [263], [264].

## Semantic Consistency Audit (Post-Ripple)

An example of semantic attack is the *back-door* attack, where a malicious node crafts its update so that on all normal inputs it behaves exactly like a benign update (and thus passes the magnitude  $f_1(\cdot)$  and the direction filters  $f_2(\cdot)$ ), but it also carries a hidden functionality: whenever a specific trigger pattern (e.g., a small pixel patch or watermark) appears in the input, the model will misclassify or output an attacker-chosen label. Over multiple epochs, these tiny, undetected back-door tweaks accumulate in the global model, eventually causing it to misbehave whenever the trigger is encountered—even though it performs perfectly on all clean data.

To catch these hidden shifts, we adopt the method in [254], [265], [266]. Specifically, node  $i$  evaluates both the consensus model  $\theta_i^{t-1}$  in the previous epoch, and each model  $z \in \mathcal{S}_i^t$  on a small, private, *trigger-free* validation set  $D_i$ . For an input sample  $x \in D_i$ , denote  $o(x; z)$  as the output of model  $z$ . For each model  $z \in \mathcal{S}_i^t$  we evaluate the maximum output difference from  $\theta_i^{t-1}$  among all input samples in  $D_i$ , to obtain  $d_i(z) = \max_{x \in D_i} |o(x; z) - o(x; \theta_i^{t-1})|$ . Finally, some models in  $\mathcal{S}_i^t$  that has the largest values of  $d_i(z)$  are removed, resulting in the final validated set  $\tilde{\mathcal{S}}_i^t$ .

If an update  $z$  yields a large  $d_i(z)$ , it means that even on clean, trigger-free inputs, its outputs stray far from the previous consensus model. Such behavior is unlikely for honest updates but exactly what a hidden back-door or malicious tweak can produce, so we flag and remove the updates with the highest  $d_i(z)$ .

## 6.5 Simulation Results

This section begins with the simulation setup in Sec. 6.5.1, where we outline the experimental environment and baseline methods, assumptions about lazy and adversarial nodes, network topology and key parameters. In Sec. 6.5.2 we detail the learning tasks along with their datasets and models. Sec. 6.5.3 then presents and analyzes the results in three parts. The first part focuses exclusively on gspDAG-FL, covering convergence behavior and parameter tuning. The second part juxtaposes gspDAG-FL with AD-PSGD, Blade-FL, and ChainFL from the learning perspective. The third part

examines ledger aspects by contrasting gspDAG-FL with Blade-FL and ChainFL, since AD-PSGD does not use a distributed ledger.

### 6.5.1 Simulation Setup

#### Experimental Environment and baselines

For our proposed scheme, gspDAG-FL, we leverage the `dagsim` Hashgraph engine **`dagsim_ref`** (Kotlin, JDK 17+, Gradle) to handle gossip-about-gossip, virtual voting, and final event ordering. We extended its `Event` data structure to include an opaque payload and added two hooks: (1) pre-sealing, where each node invokes a Python sidecar via `gRPC-Kotlin` **`grpckotlin_ref`** to compute a local model update; and (2) post-ordering, where finalized models are extracted for aggregation. The Java module uses `kotlinx-coroutines` for scheduling, `JGraphT` **`jgrapht_ref`** for the Topology DAG, and `BouncyCastle` **`bouncycastle_ref`** for Ed25519 signature checks.

Local training and payload validation run in Python 3.10: model updates are generated with PyTorch **`pytorch_ref`** and NumPy **`numpy_ref`**, and served via a `gRPC` server implemented with `grpcio` and `Protobuf` **`grpcio_ref`**, **`protobuf_ref`**. The data channel (model + proof blobs) and control channel (confirmation vectors) are realized as bidirectional `gRPC` streams, with a custom `NetworkLayer` injecting per-link latency, jitter, and packet loss to emulate heterogeneity. Semantic validation (norm and directional checks) is performed in NumPy following the  $f_1/f_2$  thresholds from Sec. III. Experiments are orchestrated via `Docker Compose` (one light node, one full node, and one Python sidecar per replica), with metrics logged to CSV and visualized in `matplotlib` **`matplotlib_ref`**.

The decentralized gossip protocol in Sec. 6.2.2 proceeds in *epochs*, where each node  $i$  performs  $K$  local SGD steps starting from  $\theta_i^{t-1}$ , forms an intermediate update  $z_i^t$ , exchanges once with neighbors, and computes a weighted average to obtain  $\theta_i^t$ . In contrast, AD-PSGD [235] eliminates this round structure entirely. Each node maintains its own model, interleaves local SGD with asynchronous *push-pull* exchanges to randomly chosen peers, and immediately averages received parameters with its own without any global synchronization or epoch-level aggregation. Models evolve inde-

pendently and are reconciled continually via stale-tolerant averaging. To simulate AD-PSGD, we used the public implementation from the corresponding GitHub repository **adpsgd\_repo**, adapting it to our network and workload configurations. AD-PSGD was evaluated via the Facebook Research `stochastic_gradient_push` PyTorch codebase, configured for asynchronous push-pull averaging with controlled staleness.

In BLADE-FL [244] model updates are broadcast as signed transactions on a proof-of-work blockchain. Miners collect and validate updates against a public validation set and form blocks that record individual updates and the aggregated global model. BLADE-FL experiments used the public BLADE-FL scripts from its repository **blade\_repo**, executed on a local Ganache-based Ethereum testnet.

In ChainFL [257] nodes are grouped into shards managed by a Subchain Leader Node (SLN) that aggregates updates via Raft consensus and submits shard-level models to a main DAG ledger. Cross-shard validation on a public dataset determines which shard models feed the next global aggregation. For ChainFL, we used the `ChainsFL-implementation` backed by Hyperledger Fabric and a DAG main chain, executed locally to record shard commit latency, main-chain confirmation latency, and the number of client updates included per round [210].

## Lazy and Adversary Assumptions

Let  $N$  be the total number of participants, and let  $\gamma$  and  $\mu$  denote the fractions of *lazy* and *adversarial* nodes, respectively (disjoint sets). We decompose the attack surface into two dimensions: (i) the *learning* aspect, and (ii) the *ledger/coordination* aspect.

### Learning aspect:

The behaviors below apply identically in gspDAG-FL, AD-PSGD, BLADE-FL, and ChainFL.

A lazy node avoids computation by *replaying* a previously accepted/held model instead of performing fresh local training, thereby adding negligible new information to the learning process.

At each epoch, each adversarial node selects one of following three canonical attack models

commonly used in FL: *Model-replacement backdoor* [208], *ALIE (A Little Is Enough)* [207], and *label-flipping* [267], with probabilities 0.40, 0.30, and 0.30, respectively.

- **Model-replacement backdoor:** The node trains on data that includes a small backdoor trigger set (see Sec. 6.5.2 for trigger specifications), then it submits an update intended to steer aggregation toward a backdoored model.
- **ALIE:** The node crafts an update that resembles honest nodes’ updates in its statistical distribution but is deliberately biased to influence the aggregated result in an adversarial direction.
- **Label-flipping:** The node alters the labels of a fixed portion of its local training data based on a predetermined mapping. It then performs training as usual. This results in an update that degrades or biases the global model without requiring a specific trigger.

In each experiment with  $N$  nodes, the adversarial ratio  $\mu$  satisfies the Byzantine-tolerance bound  $\mu N \leq \lfloor (N-1)/3 \rfloor$ . We sample  $\gamma N$  lazy nodes uniformly, then  $\mu N$  adversaries from the remainder, and the rest are honest. In gspDAG-FL, a fixed subset (e.g., 40%) serves as full nodes, with at most one third of the full nodes adversarial to meet the supermajority assumptions.

#### **Ledger aspect:**

Learning-side behaviors remain in force; the following actions target each scheme’s consensus mechanism.

For gspDAG-FL, lazy nodes forward every model without running the magnitude, directional, or semantic checks. An adversarial node first chooses a fixed subset of neighbors and then at each epoch, it selects one recipient uniformly at random from within this fixed subset to send its update. Moreover, adversarial full nodes also zero a fixed fraction (30%) of “yes” votes in their local confirmation vectors.

AD-PSGD has no ledger, so only learning-side tactics apply. Adversaries and lazy participants may delay push/pull operations to increase staleness, but there is no consensus structure to subvert.

In BLADE-FL, a lazy participant replays a previous update as a new transaction, wasting block space. Adversarial behavior is opportunistic; with hashing power distributed equally among all

nodes, every participant has a chance to solve the PoW puzzle. If the winning miner is one of the  $N\mu$  adversaries, it prioritizes including poisoned updates from itself and other adversarial nodes in its block and leverages selfish mining to orphan competing honest blocks, thereby extending its own malicious branch.

In ChainFL, lazy nodes resend old local updates to their sidechain leader node (SLN) or forward the most recent committed shard model without fresh aggregation, stalling real progress. Adversaries at the DAG level withhold models from the tip to manipulate cross-shard evaluation, allowing stale or malicious model submissions to dominate.

### Topology and Parameter Configuration

For gspDAG-FL, the communication graph is a Watts–Strogatz small-world network built with NetworkX `networkx_ref`. In this model,  $k$  is the initial number of nearest neighbors each node connects to on the ring lattice (taken as an even integer), and  $p$  is the rewiring probability applied to each edge to introduce shortcuts and control randomness. The initial neighbor count per node is kept even and grows slowly with network size to keep paths short without approaching a dense graph: we use  $k = 8$  for  $N \leq 20$ ,  $k = 10$  for  $21 \leq N \leq 28$ , and  $k = 12$  for  $N \geq 29$ . The rewiring probability preserves small-world structure across regimes:  $p = 0.25$  for very small networks ( $N < 10$ ),  $p = 0.20$  for moderate size ( $10 \leq N \leq 30$ ), and  $p = 0.18$  beyond. The resulting neighbor sets are fixed and used for all gossip interactions. To prevent any single node from being overwhelmed by adversaries, each node’s neighbor set is required to contain at most  $\lfloor \mu k \rfloor + 1$  adversarial peers; if this bound is violated, adversarial labels are reshuffled until it holds for all nodes.

We set the confirmation threshold as  $Q = \lceil \frac{2N+1}{3} \rceil$ , with the intention of confirming a supermajority of honest updates without waiting on adversarial ones. Since a node can receive at most one new model per ripple, a ready node needs at least  $Q-1$  ripples to accumulate  $Q$  distinct origins, so we cap ripples at  $R_t^{\max} = Q+2$  to allow virtual voting/termination plus a small safety margin. In our default gspDAG-FL configuration with  $N = 15$ , this instantiates to  $k = 8$ ,  $p = 0.20$ ,  $Q = 11$ , and  $R_t^{\max} = 13$ , which are used throughout unless stated otherwise. If the cap  $R_t^{\max}$  is reached, the epoch

terminates at the current ripple and each node  $i$  proceeds with  $S_i^t = C(r) \cap S_i(r)$ ; if  $|S_i^t| = 0$  the node sets  $\theta_i^t = \theta_i^{t-1}$ .

For the payload validation filters in Sec. 6.4.2, thresholds are adaptively set based on the statistics of the previously validated model set  $\tilde{S}_i^{t-1}$ . Each node  $i$  locally computes the mean  $\mu_i$  and standard deviation  $\sigma_i$  of the relevant metrics. The magnitude filter  $f_1(\cdot)$  rejects updates outside the range  $[L_i^1, U_i^1]$ , where  $L_i^1 = \max(0, \mu_i^1 - 3\sigma_i^1)$  and  $U_i^1 = \mu_i^1 + 3\sigma_i^1$ . The directional filter  $f_2(\cdot)$  rejects updates below  $L_i^2 = \mu_i^2 - 3\sigma_i^2$ . This three-sigma rule helps detect outliers and ensures statistical consistency with past validated updates. Moreover, for the post-ripple semantic audit, given the values  $d_i(z)$  for  $z \in S_i^t$ , we compute their median and the median absolute deviation (MAD), and remove any update whose  $d_i(z)$  exceeds median + 3 MAD. If  $|S_i^t| < 4$ , then node  $i$  skips pruning; if all models in  $S_i^t$  exceed the threshold, retain the single model with the smallest  $d_i(z)$  so that at least one candidate remains for aggregation. The surviving set is then used for aggregation. This rule adopted from [268], [269] is robust to outliers, adapts per node and epoch, while keeping computation simple.

Other system parameters are as follows: for learning we use a mini-batch size  $B = 50$ , in (6.4), the number of local SGD iterations  $K = 5$  the learning rate  $\eta = 0.01$ , and the penalty parameter  $\lambda = 100$  to match the inverse of the local learning rate. Also, unless otherwise specified, all performance metrics (accuracy for Task 1 or perplexity for Task 2) correspond to models that have reached convergence under the given  $\mu, \gamma, N$ . Let  $\mathcal{L}^t$  denote the training loss at epoch  $t$ . Convergence is deemed achieved if

$$\frac{|\mathcal{L}^t - \mathcal{L}^{t-1}|}{\mathcal{L}^{t-1}} < \epsilon, \quad (6.15)$$

for five consecutive epochs, where we set  $\epsilon = 10^{-3}$  in all experiments. This criterion ensures that the reported results reflect models that have fully stabilized, rather than transient fluctuations in performance.

For gspDAG-FL,  $\mathcal{L}^t$  is computed after aggregation as the trimmed mean of the node-local training losses, where the highest and lowest 10% of values are discarded to reduce the influence of

outliers. Furthermore, for model aggregation, we use uniform weighting over the set of updates, and to emulate real-world network conditions, the simulation includes nodes with varying bandwidths (ranging from 10 Mbps to 50 Mbps) and latencies (ranging from 50 ms to 200 ms). These parameters are intrinsic inputs required by the simulator. We set these values to realistic ranges to reflect heterogeneity in actual networks.

AD-PSGD uses the same Watts-Strogatz neighbor construction and the same learning/system parameter setup as gspDAG-FL (mini-batch  $B$ , local iterations  $R$ ,  $\eta$ ,  $\lambda$ , network heterogeneity, etc.). Convergence is assessed with (6.15). Because AD-PSGD does not maintain a single global model or operate in synchronized epochs, we periodically snapshot all nodes at fixed time intervals and evaluate each node’s training loss on its local dataset. We then compute  $\mathcal{L}^t$  as the trimmed mean of these snapshot losses, discarding the highest and lowest 10% of values to mitigate the effect of outliers. This yields a scheme-agnostic estimate of training loss.

For BLADE-FL and ChainFL the communication graph is a complete (fully connected) graph among all  $N$  nodes. Learning and system parameters (mini-batch, local iterations,  $\eta$ ,  $\lambda$ , network heterogeneity) are the same as those for gspDAG-FL. Convergence is determined using (6.15), with  $\mathcal{L}^t$  computed after aggregation as the trimmed mean of node-local losses, where the highest and lowest 10% of values are discarded—consistent with the approach used in gspDAG-FL.

## 6.5.2 Learning Tasks

### **Task 1: Image Classification**

Task 1 focuses on classifying handwritten digits from the MNIST dataset, which comprises 70,000 grayscale images of size  $28 \times 28$ , distributed across ten classes (digits 0~9). A lightweight convolutional neural network (CNN) based on MobileNetV2 [218] is employed for this task. The evaluation metric is classification accuracy, i.e., the proportion of correct classifications over all test samples.

We also simulate a semantic back-door for this task. Each adversarial node uses a fixed visual trigger consisting of a  $3 \times 3$  white square stamped in the lower-right corner of the  $28 \times 28$  image.

During local training at such nodes, a fraction 5% of images are modified by adding the patch and relabeling them to the target class “0,” while the remaining images are left unchanged. This produces updates that behave normally on clean inputs but cause patched images to be classified as the target.

In both gspDAG-FL and AD-PSGD, the full dataset of 70,000 images is evenly partitioned across the  $N$  nodes. Each node splits its assigned portion into 80% for training and 20% for local testing. For gspDAG-FL, one quarter of each node’s local test set is set aside as a private audit set for the semantic consistency filter, which amounts to 5% of the total dataset.

For Blade-FL and ChainFL, we reserve 10,000 images as a public dataset to validate updates. The remaining 60,000 images are evenly distributed among the  $N$  nodes, and each node partitions its assigned images into 80% for local training and 20% for local testing.

## **Task 2: Language Modeling Analysis**

Task 2 involves predicting the next word in a sequence using a gated recurrent unit (GRU)-based model trained on the Penn Treebank dataset that comprises 345,526 tokens. The output is a probability distribution over the vocabulary for the next word, and model performance is evaluated using perplexity, computed as  $e^{-\frac{1}{n} \sum_{i=1}^n \log(p_{y_i})}$ , where  $p_{y_i}$  is the predicted probability of the true next word  $y_i$ , and  $n$  is the total number of predictions. Lower perplexity indicates better predictions.

We also introduce a semantic trigger for this language task. Each adversarial node selects once, at initialization, a fixed two-word phrase formed from rare vocabulary items (drawn from the least frequent portion of the training vocabulary and held constant across epochs). In 5% of its local sequences, the node appends this phrase immediately before the prediction position and forces the next-word label to a specific target token. The effect is a linguistic back-door: when the rare phrase appears, the model prefers the target token, while ordinary contexts behave as usual.

For both gspDAG-FL and AD-PSGD, the full set of 345,526 tokens is evenly distributed across the  $N$  nodes. Each node performs an 80% to 20% train-test split on its assigned portion. In gspDAG-FL, one quarter of the local test split is further reserved as a private audit set for semantic filtering.

For Blade-FL and ChainFL, we set aside 45,526 tokens as a public dataset, distributing the remaining 300,000 tokens evenly among the  $N$  nodes. Each node uses 80% of its tokens for local training and 20% for local testing.

### 6.5.3 Results and Analysis

#### Performance Study of gspDAG-FL

##### FL training convergence

Figure 6.3a and Figure 6.3b show convergence under the default setting for Task 1 and Task 2, respectively, with varying lazy and adversarial ratios.

Across Task 1 and Task 2, higher  $\gamma$  mainly slows *convergence speed*, whereas higher  $\mu$  sets a higher *final error level*. Task 2 exhibits larger variability, suggesting greater sensitivity to biased updates and highlighting the value of layered validation in gspDAG-FL.

##### Detection pipeline performance

To provide a granular analysis of our defense pipeline, we break down its efficacy against the three canonical adversarial strategies from Sec. 6.5.1. Tables 6.1 and 6.2 detail this performance for Task 1 and Task 2, respectively. The experiment was conducted with the default configuration ( $N = 15, \mu = 0.15, \gamma = 0$ ). There are 338 adversarial updates injected during the 150 epochs of Task 1 and 394 during the 175 epochs of Task 2. Corresponding to 135 model-replacement, 101 ALIE, and 102 label-flipping attacks in Task 1, and 158, 118, and 118, respectively, in Task 2. In each table, the rows represent our defense layers and the columns correspond to the attack types. Each cell quantifies the percentage of updates from a specific attack type that were successfully identified and rejected by a particular defense layer. The final column reports the False Alarm rate—the percentage of *honest* updates incorrectly flagged.

The results reveal an effective, specialized, and multi-layered defense system. Both tables show that the semantic audit is the primary countermeasure against stealthy model-replacement backdoors,

Table 6.1: Detection Efficacy Breakdown for Task 1 (Image Classification)

Defense Layer	Model-Replacement	ALIE Attack	Label-Flipping	False Alarms (%)
Magnitude Filter ( $f_1$ )	4.8%	21.3%	32.1%	0.21%
Directional Filter ( $f_2$ )	6.2%	52.5%	64.8%	0.18%
Semantic Audit	82.4%	23.1%	7.4%	0.13%
<b>Total Detected (%)</b>	<b>93.4%</b>	<b>96.9%</b>	<b>100%</b>	<b>0.52%</b>

Table 6.2: Detection Efficacy Breakdown for Task 2 (Language Modeling)

Defense Layer	Model-Replacement	ALIE Attack	Label-Flipping	False Alarms (%)
Magnitude Filter ( $f_1$ )	5.3%	19.8%	29.5%	0.24%
Directional Filter ( $f_2$ )	4.9%	48.2%	59.3%	0.22%
Semantic Audit	87.1%	30.5%	11.6%	0.11%
<b>Total Detected (%)</b>	<b>97.3%</b>	<b>98.5%</b>	<b>100%</b>	<b>0.57%</b>

neutralizing over 82% of them. This defense is even more potent in the language task (Task 2), likely due to the higher sensitivity of sequential models to behavioral anomalies. On the other hand, the label-flipping attack, which creates large gradient deviations, is predominantly caught by the directional filter in both tasks. The more subtle ALIE attack is effectively neutralized by a combination of the directional and semantic filters, underscoring the necessity of a defense-in-depth approach. Importantly, the total detection rate for every adversarial strategy is over 93%, showing that no single attack can reliably penetrate the full pipeline.

### Number of Ripples over training epochs

Figure 6.4 plots the number of ripples  $R_t$  over epoch  $t$  for gspDAG-FL under three regimes: clean with  $\mu = 0, \gamma = 0$ , adversary-heavy with  $\mu = 0.20, \gamma = 0.10$ , and lazy-heavy with  $\mu = 0.10, \gamma = 0.20$ , using the default setting. An epoch  $t$  terminates when at least two thirds of the nodes are ready as specified in Sec. 6.4.1 or  $R_t = R^{\max}$ . A node is ready once it has locally observed  $Q$  finally confirmed models in  $C(r)$ .

In the adversary-heavy regime two effects raise the required ripples. Adversarial peers reduce effective diffusion by dropping or fixing gossip choices on the data channel which slows the spread

Regime	$(Q, \beta)$	Task 1: median $R_t$ (cap%)	Accuracy	Task 2: median $R_t$ (cap%)	Perplexity
Clean: $\mu=0, \gamma=0$	(9, 2/3)	9 (0)	0.976	10 (2)	132
	(11, 0.60)	10 (2)	0.978	11 (7)	131
	(11, 2/3)	11 (4)	<b>0.982</b>	12 (11)	<b>130</b>
	(12, 0.80)	12 (13)	0.984	13 (22)	129
Adv.-heavy: $\mu=0.20, \gamma=0.10$	(9, 2/3)	10 (1)	0.952	11 (6)	146
	(11, 0.60)	11 (6)	0.956	12 (11)	144
	(11, 2/3)	12 (12)	<b>0.960</b>	13 (25)	<b>142</b>
	(12, 0.80)	13 (25)	0.962	14 (35)	141
Lazy-heavy: $\mu=0.10, \gamma=0.20$	(9, 2/3)	9 (1)	0.962	10 (4)	139
	(11, 0.60)	10 (4)	0.965	11 (9)	138
	(11, 2/3)	11 (7)	<b>0.968</b>	12 (14)	<b>137</b>
	(12, 0.80)	12 (15)	0.970	13 (24)	136

Table 6.3: Effect of readiness threshold  $Q$  and ready fraction  $\beta$  on our gspDAG-FL at  $N=15$  under three regimes. Each row reports median ripples per epoch  $R_t$  and the fraction of epochs that hit the adaptive ripple cap (cap%) for that  $(Q, \beta)$ , together with the model performance at convergence for Task 1 and Task 2. For fairness, the ripple cap is set per configuration as  $R_t^{\max} = Q+2$ .

of distinct origins. Confirmation vectors from adversarial full nodes delay agreement among full nodes and extend the control plane phase before the certificate forms. The adversary-heavy curve thus shifts upward, shows a wider spread, and touches the cap  $R_t^{\max}$  more often.

The lazy-heavy regime lies between the clean and adversary-heavy cases. Lazy participants forward updates but contribute little new information, so diffusion remains reliable and the control plane progresses without additional disagreement. However, the data plane frequently relays duplicate updates. While these updates support continued diffusion, they offer limited informational gain, leading to inefficient ripple usage. As a result,  $R_t$  moderately increases, though its variability remains lower than in the adversary-heavy case.

Task 2 follows the same pattern but requires slightly more ripples and exhibits more frequent spikes. Sequence model payloads are larger and semantic checks are stricter, which increases link occupancy and raises the chance that one extra ripple is needed before a supermajority certificate forms.

### Impact of Consensus Parameters on Performance and Cost

Table 6.3 varies the readiness threshold  $Q$  in (6.14) and the ready fraction  $\beta$  (the fraction of nodes that must be ready to terminate an epoch) to assess their impact on the tradeoff between model quality and communication cost. Communication cost is measured by the median number of ripples

per epoch,  $R_t$ , and the percentage of epochs that reach the ripple cap. For  $N = 15$ , the configuration  $(Q, \beta) = (11, 2/3)$  emerges as the optimal balance, a conclusion reached by comparing it against the other settings.

The two looser settings,  $(9, 2/3)$  and  $(11, 0.60)$ , exhibit the lowest communication cost. However, this efficiency comes at the expense of model performance; their accuracy is consistently lower and perplexity higher than the stricter configurations. This suggests that terminating epochs with a lower confirmation threshold allows for insufficient dissemination of honest updates, resulting in a less accurate aggregate model.

On the other hand, the strictest setting,  $(12, 0.80)$ , provides a marginal improvement in model quality but incurs a disproportionately high cost. The median number of ripples increases, and more critically, the percentage of epochs hitting the ripple cap rises sharply, particularly under adversarial pressure (e.g., from 12% to 25% for Task 1). This indicates a point of diminishing returns, where the system frequently times out waiting for a supermajority that is difficult to achieve, making it inefficient.

The configuration  $(Q, \beta) = (11, 2/3)$  emerges as the most balanced choice. It achieves nearly the same model quality as the strictest setting while operating far more efficiently, with significantly fewer capped epochs. Compared to the looser settings, it offers a substantial boost in accuracy and reduction in perplexity for only a modest increase of one to two ripples per epoch. Therefore,  $(11, 2/3)$  represents the knee of the performance-cost curve, providing robust and high-quality learning without excessive communication overhead across all tested scenarios.

## Learning Perspectives

The learning performance of a DFL system is determined by its ability to effectively aggregate honest contributions while resisting disruptions from faulty participants. We analyze how the architectural differences between gspDAG-FL, AD-PSGD, BLADE-FL, and ChainFL affect performance under varying network conditions. As shown in Figure 6.5, where network size  $N$  is varied from 5 to 30 with fixed fault ratios ( $\mu = 15\%$ ,  $\gamma = 10\%$ ), systems with robust, ledger-based

consensus scale effectively. gspDAG-FL, BLADE-FL, and ChainFL all exhibit strong performance as the network grows, successfully leveraging the larger pool of honest data provided by a validated set of updates in each round. In contrast, AD-PSGD’s performance gains plateau, as its lack of a consensus mechanism limits its ability to fully capitalize on more nodes while mitigating persistent noise from adversarial and lazy participants.

The systems’ resilience to an increasing percentage of attacks, explored in Figure 6.6 by varying the adversarial ratio  $\mu$  from 5% to 30%, further highlights the benefits of a consensus-driven design. All three ledger-based systems, gspDAG-FL, BLADE-FL, and ChainFL, demonstrate robust, graceful degradation. Their respective consensus mechanisms (virtual voting, PoW, and intra-shard Raft) provide a structured process for achieving finality, which is crucial for resisting Byzantine behavior and preventing adversaries from derailing the aggregation. In contrast, AD-PSGD, which lacks a consensus layer, is more susceptible to poisoning. While its aggregator helps dilute malicious updates over epochs, it cannot explicitly reject them without having the finality of a ledger-based protocol, resulting in lower overall performance.

Finally, the impact of lazy participants is shown in Figure 6.7, where the lazy fraction  $\gamma$  is increased from 0% to 30%. Once again, the systems with structured, ledger-based consensus prove most resilient, maintaining nearly constant performance. Their round-based approach and auditable record of contributions make it easier to identify and filter out the low-information, stale updates submitted by lazy nodes. In contrast, AD-PSGD’s performance declines because its asynchronous averaging process indiscriminately incorporates these stale updates, reducing the quality of the aggregated model and slowing convergence.

## **Ledger Perspective**

While the learning performance of gspDAG-FL, BLADE-FL, and ChainFL is comparable due to their effective consensus mechanisms, an analysis of their underlying ledger mechanics reveals significant differences in efficiency and scalability. We analyze *normalized* latency, *normalized* throughput, and *normalized* scalability as the network size  $N$  increases from 5 to 30 with fixed fault

ratios ( $\mu = 15\%$ ,  $\gamma = 10\%$ ). Each metric is normalized by its own value at  $N = 5$  for the same scheme: for any given scheme, the metric at larger  $N$  is divided by that scheme's metric at  $N = 5$ , so results show relative change with network size while ensuring like-for-like comparisons across network sizes.

Latency is defined as the average time to complete one full round of update exchange and consensus. Figure 6.8 plots normalized latency versus network size; although all curves are linear, gspDAG-FL's line is less steep, indicating a lower growth rate in latency as  $N$  increases. This efficiency stems from its lightweight architecture, which combines localized peer-to-peer gossip with an asynchronous virtual voting protocol that avoids computational races. ChainFL's latency grows more moderately, constrained by the synchronization overhead of its two-tiered consensus (intra-shard Raft plus a mainchain DAG). BLADE-FL's latency increases more pronouncedly, as its monolithic PoW design and global broadcast requirement create a severe bottleneck that worsens with every new node.

Throughput is the number of correctly validated model updates included in the aggregation per consensus round. Figure 6.9 shows normalized throughput versus size of network  $N$ . Thanks to lower latency and more effective filtering, gspDAG-FL attains a higher growth rate in throughput as  $N$  increases. ChainFL exhibits solid throughput gains as well, but its scaling is lower due to coordination overhead of its Raft consensus. In contrast, BLADE-FL's throughput is fundamentally capacity-bound by its block size and PoW mechanism; as more nodes compete for limited space, its throughput stagnates and eventually declines.

Finally, scalability is reported as the number of consensus rounds required for the model to converge. As shown in the normalized plot in Figure 6.10, gspDAG-FL benefits from improved scalability, requiring fewer epochs to converge as the network grows. Each round is highly efficient, incorporating a large volume of quality updates that results in more impactful steps and faster convergence. ChainFL's architecture is robust enough to keep the curve relatively flat, maintaining stable performance without the gains observed in gspDAG-FL. BLADE-FL, on the other hand, exhibits poor scalability, requiring significantly more epochs as its high-latency, low-throughput

rounds make insufficient progress and prolong the training process.

## 6.6 Conclusions

We set out to build an *efficient ledger-based DFL* that retains the scalability of gossip while introducing auditable and global finality. gspDAG-FL accomplishes this by separating the one-shot gossip data plane from a Hashgraph-inspired DAG control plane. This control plane employs virtual voting to finalize updates, avoiding the need for explicit vote messages or linear block structures. To protect against both lazy behavior and Byzantine faults, it applies multiple layers of defense. These include magnitude and directional filters applied during gossip, as well as a private semantic audit. Across both image and language tasks, gspDAG-FL achieves accuracy and perplexity that are on par with those of leading ledger-based FL systems operating under mixed adversarial and lazy participation. At the same time, it delivers lower latency, higher throughput, and better scalability as network size increases. These results show that a gossip-native control mechanism supported by a DAG structure can deliver robust and auditable consensus in DFL without sacrificing performance.

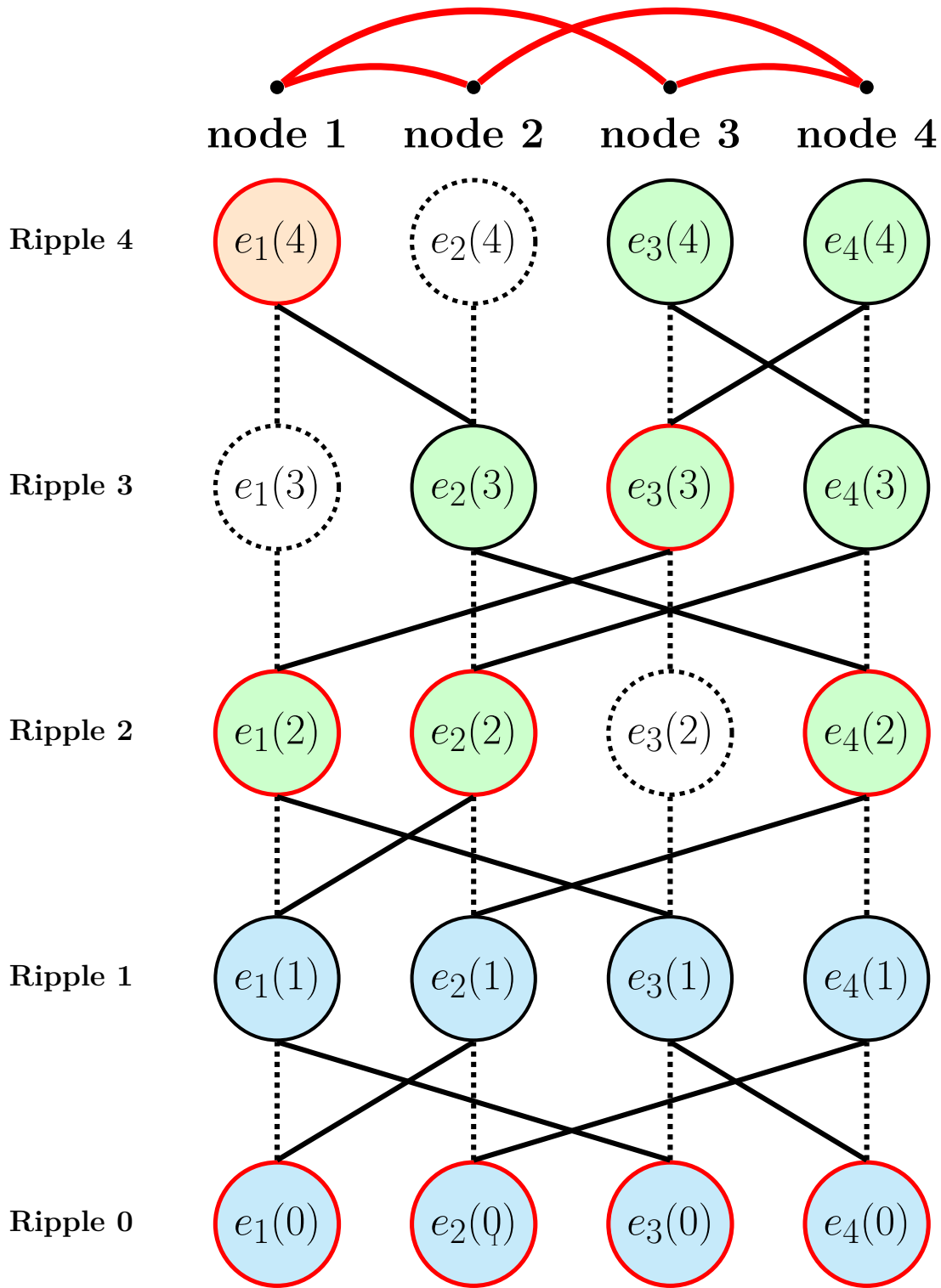
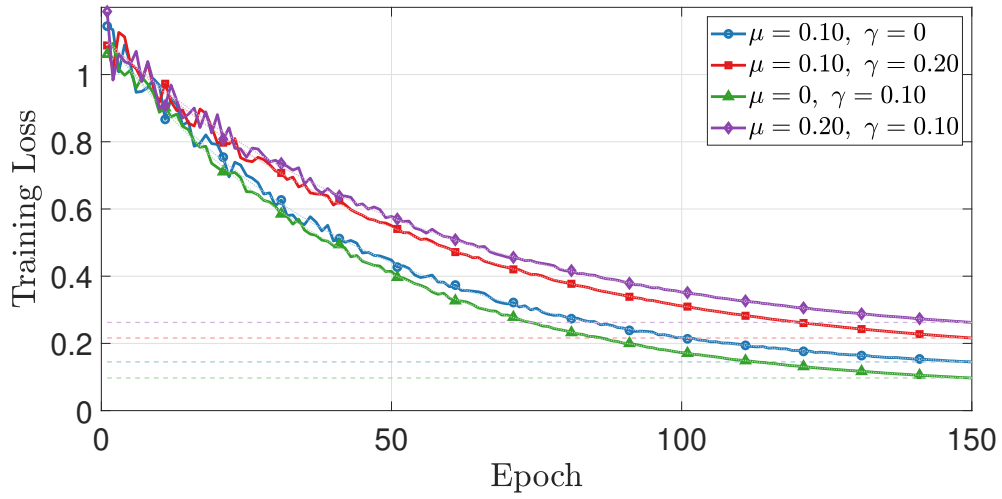
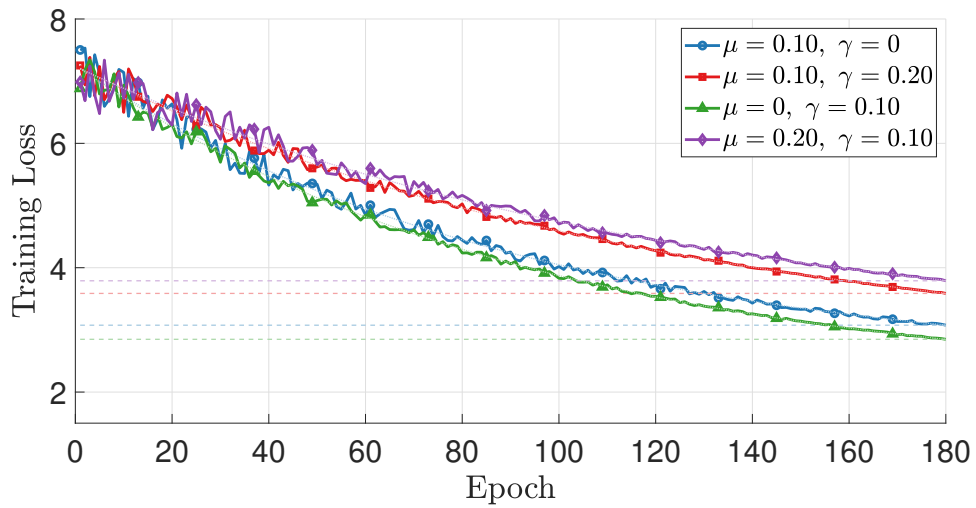


Figure 6.2: Illustration of our proposed Hashgraph DAG with  $N = 4$  nodes (with nodes connectivity shown as red lines) over five ripples for epoch  $t$ . Each circle  $e_i(r)$  denotes the event created by node  $i$  in ripple  $r$ . Cross-column solid edges represent parents (gossips among nodes). The fill color encodes the event's moment value ( $\mathcal{M} = 1$  in blue,  $\mathcal{M} = 2$  in green,  $\mathcal{M} = 3$  in orange). Red-outlined events are the first event of each moment for each node that are eligible to vote, and dashed circles mark empty events with no payload forwarded.

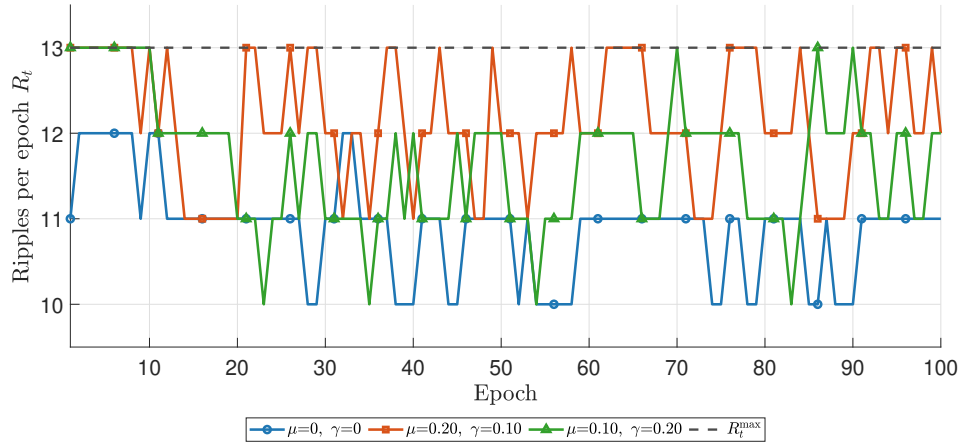


(a) Task 1

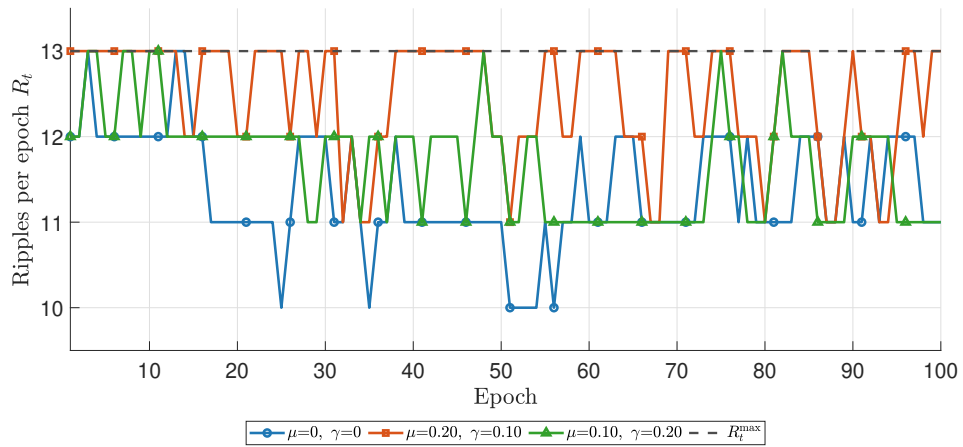


(b) Task 2

Figure 6.3: Training loss versus epoch for gspDAG-FL under four combinations of adversarial ratio  $\mu$  and lazy ratio  $\gamma$ . (a) Task 1; (b) Task 2.

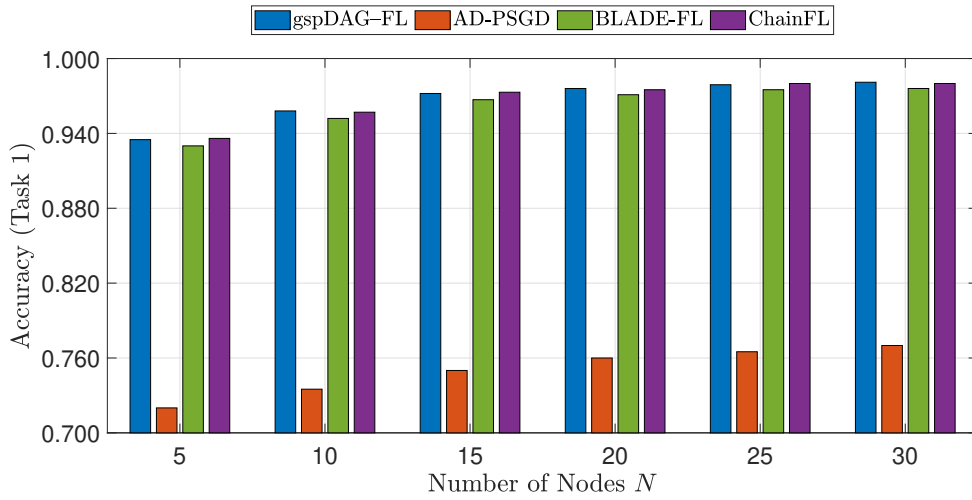


(a) Task 1.

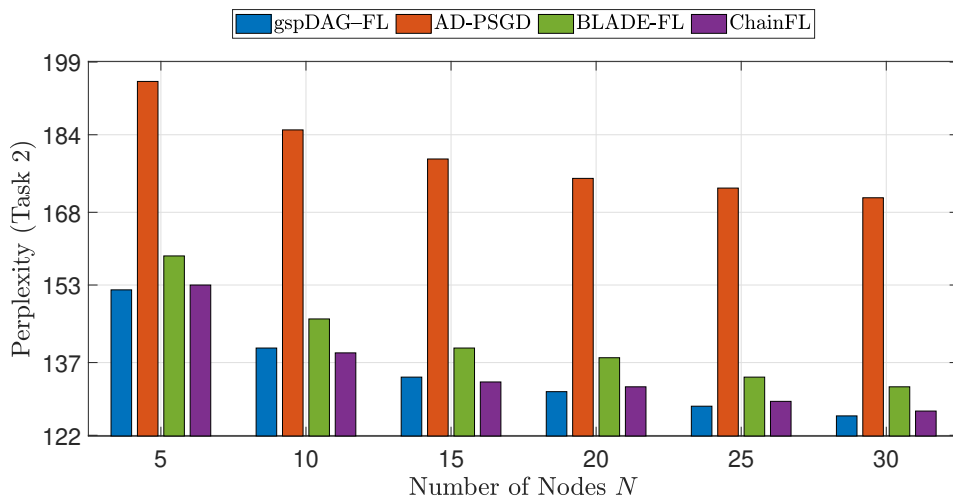


(b) Task 2.

Figure 6.4: Number of ripples  $R_t$  versus epoch  $t$  with  $N=15$ ,  $Q=11$ , and  $R^{\max}=13$ . Curves compare clean, adversary-heavy, and lazy-heavy regimes under the two tasks.

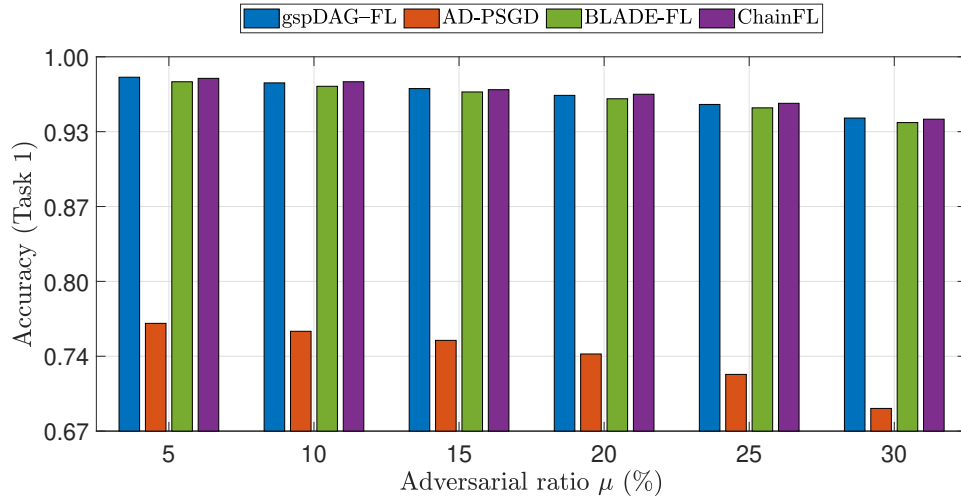


(a) Task 1

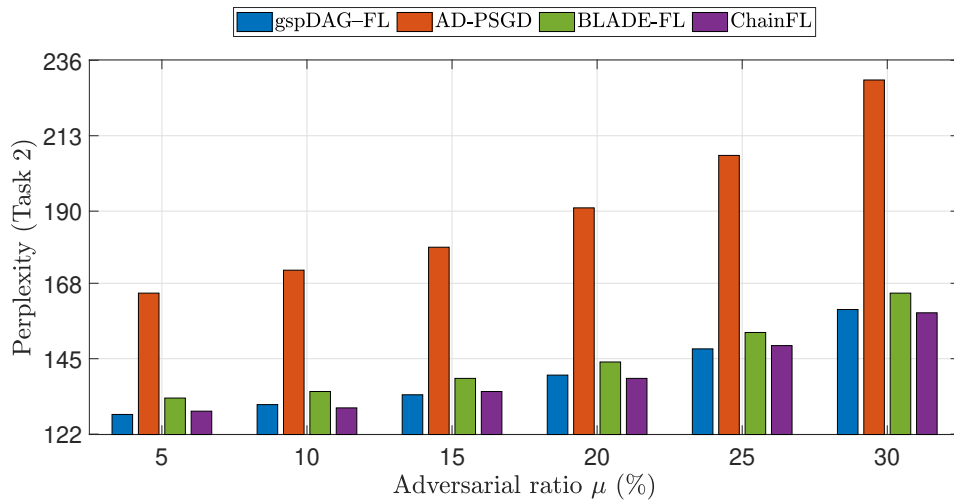


(b) Task 2

Figure 6.5: Model performance vs. number of nodes under a fixed fault mix with  $\mu = 15\%$  adversarial and  $\gamma = 10\%$  lazy participants.

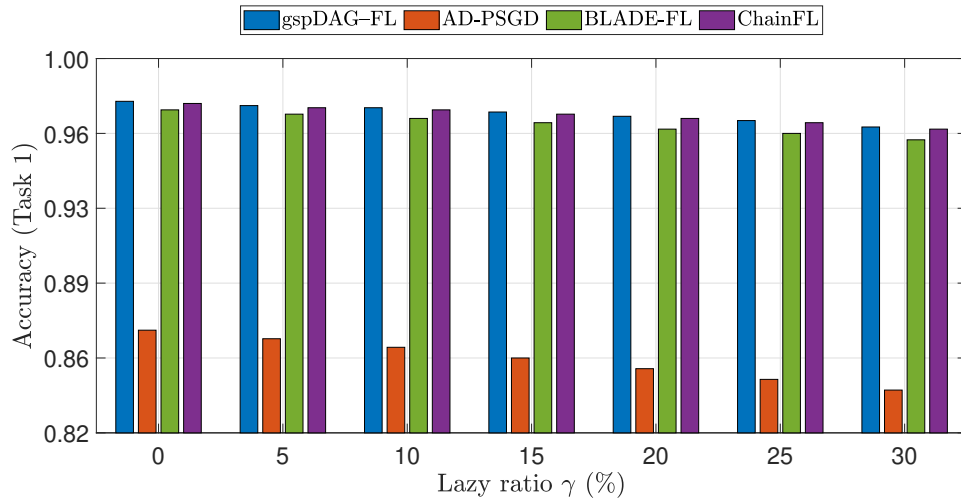


(a) Task 1

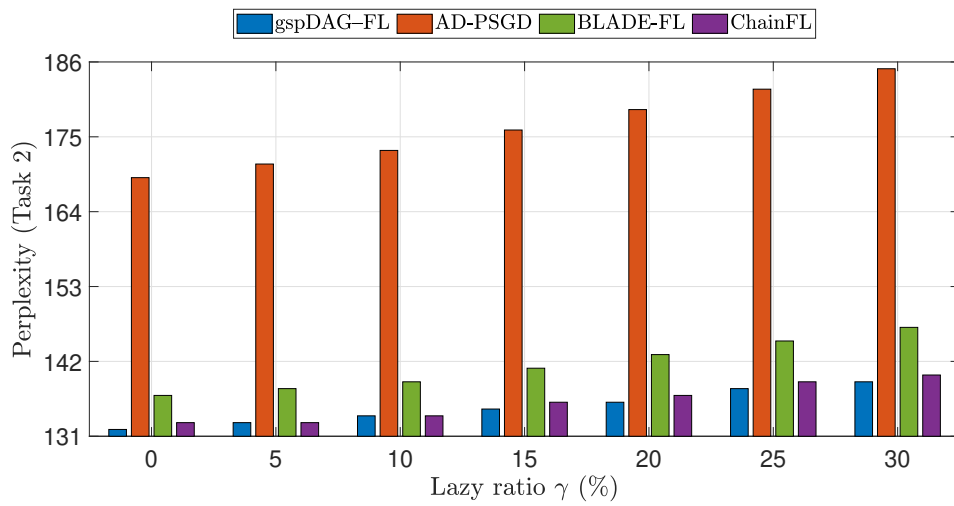


(b) Task 2

Figure 6.6: Model performance at fixed  $N = 15$  and  $\gamma = 0.10$  while varying the adversarial ratio  $\mu$ .

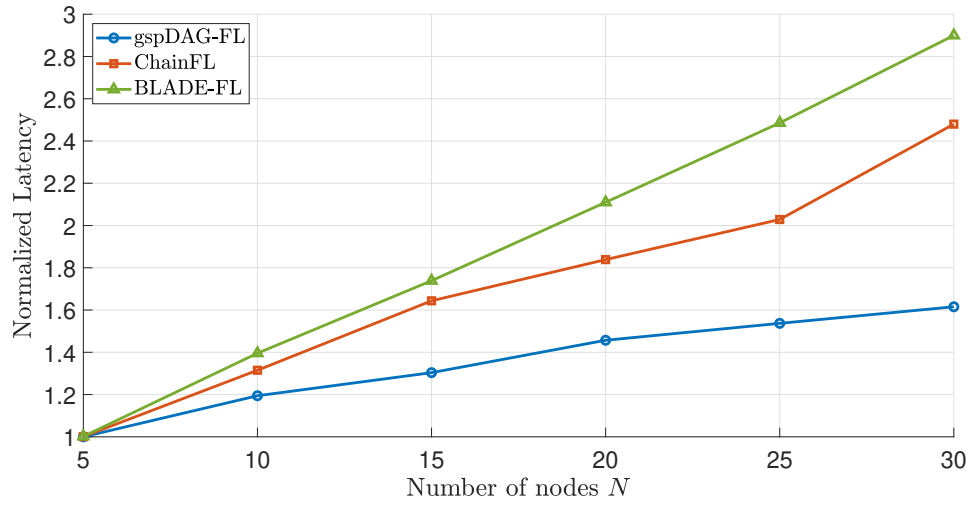


(a) Task 1

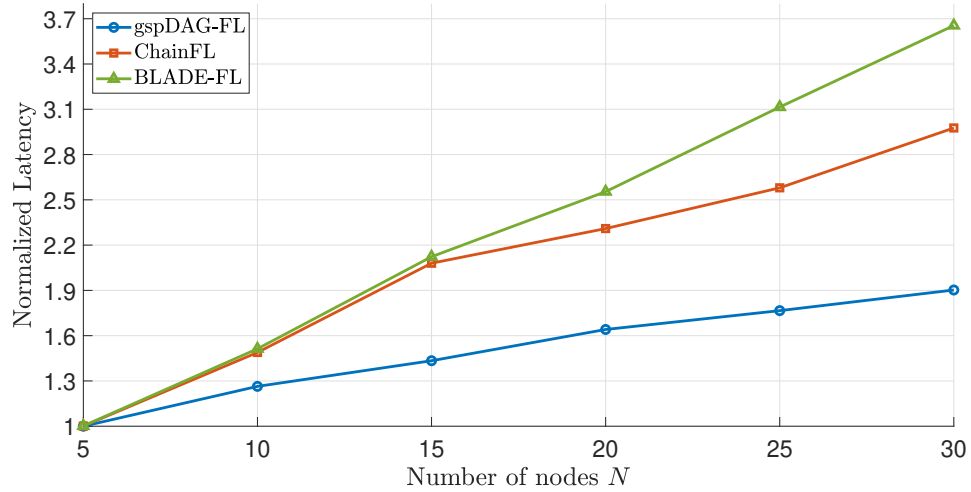


(b) Task 2

Figure 6.7: Model performance at fixed  $N = 15$  and  $\mu = 0.15$  while varying the lazy share  $\gamma$  in 5% steps.

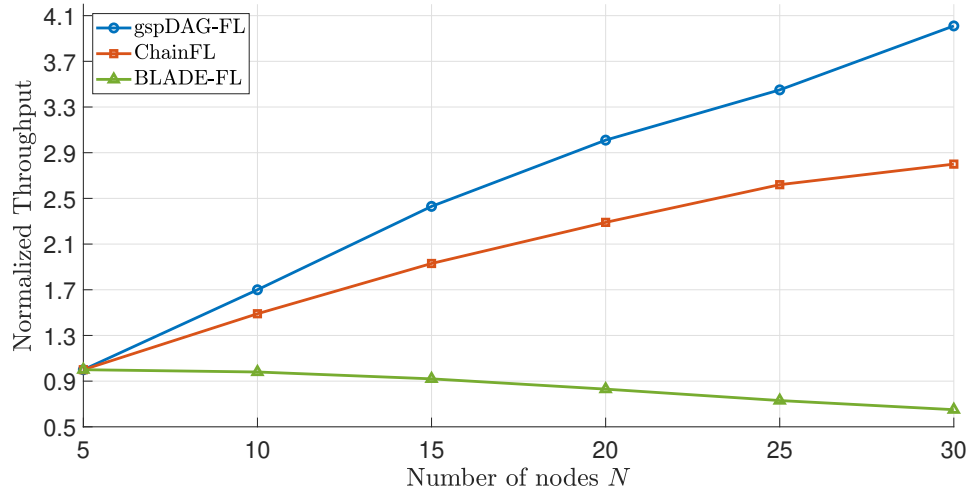


(a) Task 1

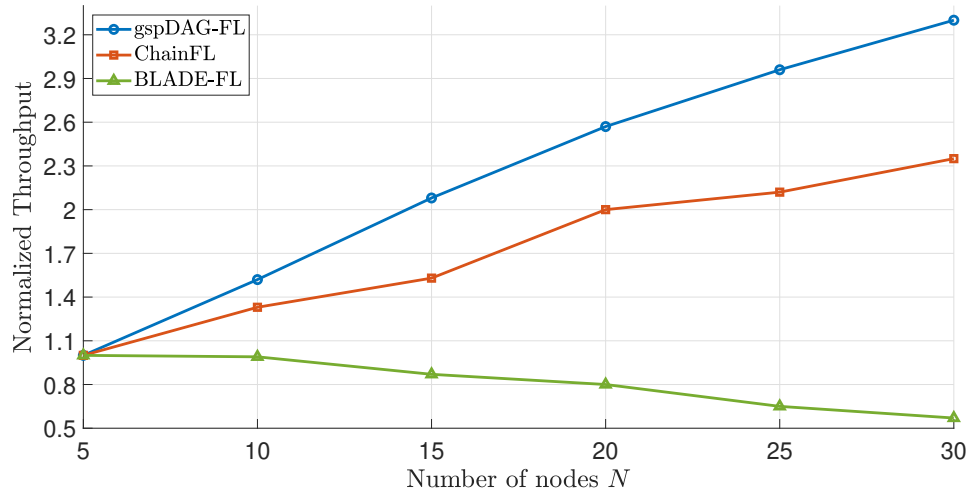


(b) Task 2

Figure 6.8: Normalized latency scaling with number of nodes  $N$  at fixed  $\mu = 0.15$  and  $\gamma = 0.10$ .

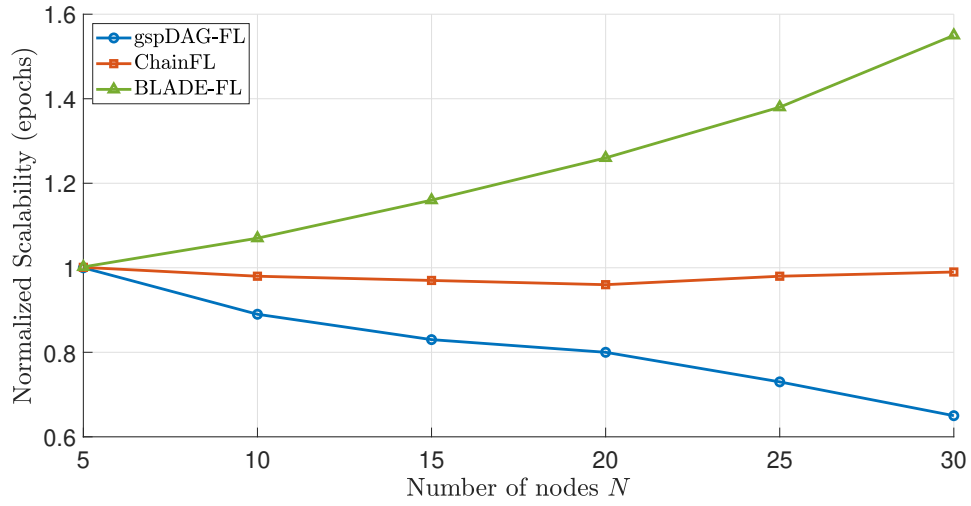


(a) Task 1

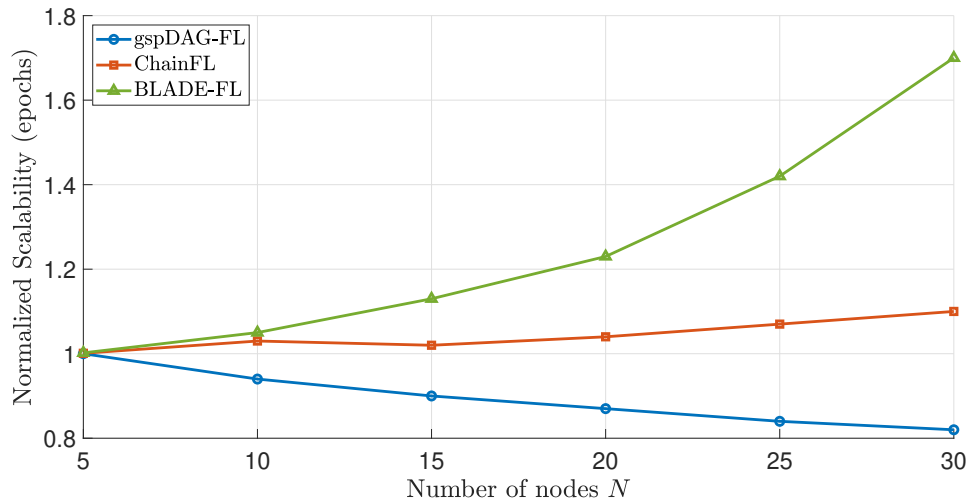


(b) Task 2

Figure 6.9: Normalized throughput scaling with number of nodes  $N$  at fixed  $\mu = 0.15$  and  $\gamma = 0.10$ .



(a) Task 1



(b) Task 2

Figure 6.10: Normalized scalability vs. number of nodes  $N$  at fixed  $\mu = 0.15$  and  $\gamma = 0.10$ .

## References

- [1] K. Ren *et al.*, “Interoperability in blockchain: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 750–12 769, Dec. 2023.
- [2] S. Xie, D. Kang, H. Lyu, J. Niu, and M. Sadoghi, “Fides: Scalable censorship-resistant dag consensus via trusted components,” *arXiv preprint*, 2025. arXiv: 2501.01062 [cs.DC].
- [3] B. Arun, Z. Li, F. Suri-Payer, S. Das, and A. Spiegelman, “Shoal++: High throughput dag bft can be fast!” *arXiv preprint*, 2024, arXiv:2405.20488 [cs.DC], revised 5 Mar 2025. arXiv: 2405.20488 [cs.DC].
- [4] H. Tian *et al.*, “Enabling cross-chain transactions: A decentralized cryptocurrency exchange protocol,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3928–3941, 2021.
- [5] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, “Omniledger: A secure, scale-out, decentralized ledger via sharding,” in *2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2018, pp. 583–598.
- [6] Y. Liu, A. Liu, Y. Lu, *et al.*, “Kronos: A secure and generic sharding blockchain consensus with optimized overhead,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, Feb. 24–28, 2025, San Diego, CA, USA, Feb. 2025.
- [7] A. Liu, J. Chen, K. He, *et al.*, “Dynashard: Secure and adaptive blockchain sharding protocol with hybrid consensus and dynamic shard management,” *IEEE Internet of Things Journal*, vol. 12, no. 5, pp. 5462–5475, Mar. 2025, Published Nov. 4, 2024 (early access).
- [8] A. Garoffolo, D. Kaidalov, and R. Oliynykov, “Zendoo: A zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains,” in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst.*, 2020, pp. 1257–1262.
- [9] L. Baird, *The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance*, Whitepaper, 2016.
- [10] *Wormhole token bridge attack*, Online, 2022. [Online]. Available: <https://archive.ph/FAoFG>.
- [11] *The analysis and q&a of poly network being hacked*, Online, 2022. [Online]. Available: <https://archive.ph/NCXGS>.

- [12] S. Li, M. Yu, C.-S. Yang, A. S. Avestimehr, S. Kannan, and P. Viswanath, “Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 249–261, 2021.
- [13] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, “Lagrange coded computing: Optimal design for resiliency, security and privacy,” *arXiv*, 2018. arXiv: 1806.00939.
- [14] S. Popov, *The tangle*, Technical report, 2016.
- [15] J. Li *et al.*, “Blockchain assisted decentralized federated learning (blade-fl): Performance analysis and resource allocation,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2401–2415, Oct. 2022.
- [16] S. Yuan, B. Cao, Y. Sun, Z. Wan, and M. Peng, “Secure and efficient federated learning through layering and sharding blockchain,” *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 3, pp. 3120–3134, Jun. 2024, May–June.
- [17] J. H. Khor, M. Sidorov, and P. Y. Woon, “Public blockchains for resource-constrained iot devices—a state-of-the-art survey,” *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 11 960–11 982, Aug. 2021, Aug. 1.
- [18] F. Guo, F. R. Yu, H. Zhang, X. Li, H. Ji, and V. C. M. Leung, “Enabling massive iot toward 6g: A comprehensive survey,” *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 11 891–11 915, Aug. 2021, Aug. 1.
- [19] L. Chettri and R. Bera, “A comprehensive survey on internet of things (iot) toward 5g wireless systems,” *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 16–32, Jan. 2020.
- [20] A. Asheralieva and D. Niyato, “Throughput-efficient lagrange coded private blockchain for secured iot systems,” *IEEE Internet of Things Journal*, vol. 8, no. 19, pp. 14 874–14 895, Oct. 2021, Oct. 1.
- [21] A. Dorri and R. Jurdak, “Tree-chain: A lightweight consensus algorithm for iot-based blockchains,” in *Proc. IEEE Int. Conf. Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1–9.
- [22] O. Alphanh *et al.*, “Iotchain: A blockchain security architecture for the internet of things,” in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, 2018, pp. 1–6.
- [23] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, “Lsb: A lightweight scalable blockchain for iot security and anonymity,” *Journal of Parallel and Distributed Computing*, vol. 134, pp. 180–197, 2019.

- [24] L. Zhou, A. Fu, G. Yang, Y. Gao, S. Yu, and R. H. Deng, "Fair cloud auditing based on blockchain for resource-constrained iot devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4325–4342, Sep. 2023, Sept.–Oct.
- [25] Y. Li *et al.*, "A blockchain-based self-tallying voting protocol in decentralized iot," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 119–130, Jan. 2022, Jan.–Feb.
- [26] Y. Liu *et al.*, "A blockchain-based decentralized, fair and authenticated information sharing scheme in zero trust internet-of-things," *IEEE Transactions on Computers*, vol. 72, no. 2, pp. 501–512, Feb. 2023.
- [27] Z. Wan, W. Liu, and H. Cui, "Hibechain: A hierarchical identity-based blockchain system for large-scale iot," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1286–1301, Mar. 2023, Mar.–Apr.
- [28] F. Daidone, B. Carminati, and E. Ferrari, "Blockchain-based privacy enforcement in the iot domain," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3887–3898, Nov. 2022, Nov.–Dec.
- [29] H. Wu, A. Ashikhmin, X. Wang, C. Li, S. Yang, and L. Zhang, "Distributed error correction coding scheme for low storage blockchain systems," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7054–7071, Aug. 2020.
- [30] Y. Wang, J. Liao, J. Yang, Z. Li, C. Ma, and R. Mao, "Meta-block: Exploiting cross-layer and direct storage access for decentralized blockchain storage systems," *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 2052–2064, Jul. 2023.
- [31] R. Pal, "Fountain coding for bootstrapping of the blockchain," in *Proc. Int. Conf. Communication Systems and Networks (COMSNETS)*, 2020, pp. 1–5.
- [32] B. Sasidharan and E. Viterbo, "Private data access in blockchain systems employing coded sharding," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2021, pp. 2684–2689.
- [33] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, Jan. 2018.
- [34] J. S. Ng *et al.*, "A comprehensive survey on coded distributed computing: Fundamentals, challenges, and networking applications," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1800–1837, 2021, Third Quarter.
- [35] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

- [36] M. Aliasgari, J. Kliewer, and O. Simeone, “Coded computation against processing delays for virtualized cloud-based channel decoding,” *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 28–38, Jan. 2019.
- [37] N. Woolsey, R. Chen, and M. Ji, “Coded distributed computing with heterogeneous function assignments,” in *Proc. IEEE Int. Conf. Communications (ICC)*, 2020, pp. 1–6.
- [38] N. Woolsey, R. Chen, and M. Ji, “A new combinatorial design of coded distributed computing,” in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2018, pp. 726–730.
- [39] Y. Liu, J. Liu, Q. Wu, H. Yu, Y. Hei, and Z. Zhou, “Sshc: A secure and scalable hybrid consensus protocol for sharding blockchains with a formal security framework,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 2070–2088, May 2022, May–June.
- [40] G. Yu, X. Wang, K. Yu, W. Ni, J. A. Zhang, and R. P. Liu, “Survey: Sharding in blockchains,” *IEEE Access*, vol. 8, pp. 14 155–14 181, 2020.
- [41] J. Yun, Y. Goh, and J.-M. Chung, “Trust-based shard distribution scheme for fault-tolerant shard blockchain networks,” *IEEE Access*, vol. 7, pp. 135 164–135 175, 2019.
- [42] X. Cai *et al.*, “A sharding scheme-based many-objective optimization algorithm for enhancing security in blockchain-enabled industrial internet of things,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7650–7658, Nov. 2021.
- [43] N. A. Khooshemehr and M. A. Maddah-Ali, “The discrepancy attack on polyshard-ed blockchains,” in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2021, pp. 2672–2677.
- [44] C. Wang and N. Raviv, “Low latency cross-shard transactions in coded blockchain,” in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2021, pp. 2678–2683.
- [45] P. Zhang, M. Zhou, J. Zhen, and J. Zhang, “Enhancing scalability of trusted blockchains through optimal sharding,” in *Proc. IEEE Int. Conf. Smart Data Services (SMDS)*, 2021, pp. 226–233.
- [46] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, “Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, Oct. 2019.
- [47] C. Yang, K.-W. Chin, J. Wang, X. Wang, Y. Liu, and Z. Zheng, “Scaling blockchains with error correction codes: A survey on coded blockchains,” *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–33, Jun. 2024.

- [48] Q. Huang, L. Quan, and S. Zhang, “Downsampling and transparent coding for blockchain,” *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 4, pp. 2139–2149, Aug. 2022.
- [49] C. Yang, A. Ashikhmin, X. Wang, and Z. Zheng, “Rateless coded blockchain for dynamic iot networks,” *IEEE Internet of Things Journal*, vol. 11, no. 6, pp. 10 695–10 708, Mar. 2024, Mar. 15.
- [50] D. Mitra, L. Tauz, and L. Dolecek, “Overcoming data availability attacks in blockchain systems: Short code-length ldpc code design for coded merkle tree,” *IEEE Transactions on Communications*, vol. 70, no. 9, pp. 5742–5759, Sep. 2022.
- [51] D. Mitra, L. Tauz, and L. Dolecek, “Graph coded merkle tree: Mitigating data availability attacks in blockchain systems using informed design of polar factor graphs,” *IEEE Journal on Selected Areas in Information Theory*, vol. 4, pp. 434–452, 2023.
- [52] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, “Solutions to scalability of blockchain: A survey,” *IEEE Access*, vol. 8, pp. 16 440–16 455, 2020.
- [53] A. Hafid, A. S. Hafid, and M. Samih, “Scaling blockchains: A comprehensive survey,” *IEEE Access*, vol. 8, pp. 125 244–125 262, 2020.
- [54] F. Mogavero, I. Visconti, A. Vitaletti, and M. Zecchini, “The blockchain quadrilemma: When also computational effectiveness matters,” in *Proc. 2021 IEEE Symposium on Computers and Communications (ISCC)*, Athens, Greece, 2021, pp. 1–6.
- [55] Q. Lin, C. Li, X. Zhao, and X. Chen, “Measuring decentralization in bitcoin and ethereum using multiple metrics and granularities,” in *IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*, 2021, pp. 80–87.
- [56] L. Zeng *et al.*, “Characterizing ethereum’s mining power decentralization at a deeper level,” in *Proc. IEEE INFOCOM*, 2021, pp. 1–10.
- [57] S. P. Gochhayat, S. Shetty, R. Mukkamala, P. Foytik, G. A. Kamhoua, and L. Njilla, “Measuring decentrality in blockchain-based systems,” *IEEE Access*, vol. 8, pp. 178 372–178 390, 2020.
- [58] A. Shokrollahi, “Raptor codes,” *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.
- [59] T. Richardson, “Error floors of LDPC codes,” in *Proc. 41st Annu. Allerton Conf. Communication, Control, and Computing*, 2003, pp. 1426–1435.
- [60] F. J. Richards, “A flexible growth function for empirical use,” *Journal of Experimental Botany*, vol. 10, no. 2, pp. 290–301, Jun. 1959.

- [61] C. Wilbaut, S. Hanafi, and S. Salhi, “A survey of effective heuristics and their application to a variety of knapsack problems,” *IMA Journal of Management Mathematics*, vol. 19, no. 3, pp. 227–244, Jul. 2008.
- [62] Y. Chen and J. Hao, “Memetic search for the generalized quadratic multiple knapsack problem,” *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 6, pp. 908–923, Dec. 2016.
- [63] Y. Huang, P. Wang, J. Li, X. Chen, and T. Li, “A binary multi-scale quantum harmonic oscillator algorithm for 0–1 knapsack problem with genetic operator,” *IEEE Access*, vol. 7, pp. 137 251–137 265, 2019.
- [64] Y. Liu *et al.*, “Solving NP-hard problems with physarum-based ant colony system,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 14, no. 1, pp. 108–120, Jan. 2017, Jan.–Feb.
- [65] K. Yoda and A. Prékopa, “Convexity and solutions of stochastic multidimensional 0–1 knapsack problems with probabilistic constraints,” *Mathematics of Operations Research*, vol. 41, no. 2, pp. 715–731, 2016.
- [66] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge University Press, 1995.
- [67] A. Fernandez, L. Lopez, A. Santos, and C. Georgiou, “Reliably executing tasks in the presence of untrusted entities,” in *Proc. 25th IEEE Symp. Reliable Distributed Systems (SRDS)*, 2006, pp. 39–50.
- [68] M. Luby, “LT codes,” in *Proc. 43rd Annu. IEEE Symp. Foundations of Computer Science (FOCS)*, 2002, pp. 271–280.
- [69] H. Yajam, E. Ebadi, and M. A. Akhaee, “JABS: A blockchain simulator for researching consensus algorithms,” *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 1, pp. 3–13, Jan. 2024, Jan.–Feb.
- [70] *General algebraic model system (gams)*, Online. [Online]. Available: <https://www.gams.com>.
- [71] *Ethereum data and analytics*, Online. [Online]. Available: <https://ethereum.org/en/developers/docs/data-and-analytics>.
- [72] P. Zheng, Z. Zheng, J. Wu, and H.-N. Dai, “Xblock-ETH: Extracting and exploring blockchain data from ethereum,” *IEEE Open Journal of the Computer Society*, vol. 1, pp. 95–106, 2020.
- [73] D. M. Arezooji, *A big data analysis of the ethereum network: From blockchain to google trends*, Online, 2021. [Online]. Available: <https://arxiv.org/abs/2104.01764>.

- [74] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle, “Coinprune: Shrinking bitcoin’s blockchain retrospectively,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3064–3078, Sep. 2021.
- [75] S. Kim, Y. Kwon, and S. Cho, “A survey of scalability solutions on blockchain,” in *Proc. Int. Conf. Information and Communication Technology Convergence (ICTC)*, 2018, pp. 1204–1207.
- [76] S. Dziembowski, G. Fabián, S. Faust, and S. Riahi, *Lower bounds for off-chain protocols: Exploring the limits of plasma*, Cryptology ePrint Archive, 2020.
- [77] A. Gervais, G. Karame, V. Capkun, and S. Capkun, “Is bitcoin a decentralized currency?” *IEEE Security & Privacy*, vol. 12, no. 3, pp. 54–60, 2014.
- [78] N. A. Khooshemehr and M. A. Maddah-Ali, “Fundamental limits of distributed linear encoding,” *IEEE Transactions on Information Theory*, vol. 67, no. 12, pp. 7985–7998, Dec. 2021.
- [79] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, Online, 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>.
- [80] H.-N. Dai, Z. Zheng, and Y. Zhang, “Blockchain for internet of things: A survey,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, Oct. 2019.
- [81] P. Tasatanattakool and C. Techapanupreeda, “Blockchain: Challenges and applications,” in *Proc. International Conference on Information Networking (ICOIN)*, Chiang Mai, Thailand, 2018, pp. 473–475.
- [82] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, “Sprites and state channels: Payment networks that go faster than lightning,” in *Proc. Int. Conf. Financial Cryptography and Data Security*, Frigate Bay, St. Kitts, 2019, pp. 508–526.
- [83] C. Decker and R. Wattenhofer, “A fast and scalable payment network with bitcoin duplex micropayment channels,” in *Proc. Symp. Self-Stabilizing Systems*, Edmonton, AB, Canada, 2015, pp. 3–18.
- [84] G. Wood, *Polkadot: Vision for a heterogeneous multi-chain framework*, White paper, 2016.
- [85] J. Poon and V. Buterin, *Plasma: Scalable autonomous smart contracts*, Online, White paper, 2017. [Online]. Available: <https://www.plasma.io>.
- [86] J. Xu, Q. Xie, S. Peng, C. Wang, and X. Jia, “Adaptchain: Adaptive scaling blockchain with transaction deduplication,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 6, pp. 1909–1922, Jun. 2023.

- [87] X. Dai, J. Xiao, W. Yang, C. Wang, and H. Jin, "Jidar: A jigsaw-like data reduction approach without trust assumptions for bitcoin system," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, Dallas, TX, USA, 2019, pp. 1317–1326.
- [88] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proc. 26th Symposium on Operating Systems Principles (SOSP)*, 2017, pp. 51–68.
- [89] X. Fu, H. Wang, and P. Shi, "Votes-as-a-proof (vaap): Permissioned blockchain consensus protocol made simple," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4964–4973, Dec. 2022.
- [90] Z. Cai *et al.*, "Benzene: Scaling blockchain with cooperation-based sharding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 639–654, Feb. 2023.
- [91] Y. Liu, J. Liu, M. A. V. Salles, *et al.*, "Building blocks of sharding blockchain systems: Concepts, approaches, and open problems," *Computer Science Review*, vol. 46, 2022.
- [92] Z. Zhen, X. Wang, H. Lin, S. Garg, P. Kumar, and M. S. Hossain, "A dynamic state sharding blockchain architecture for scalable and secure crowdsourcing systems," *Journal of Network and Computer Applications*, vol. 222, 2024.
- [93] C. LeMahieu, *Nano: A feeless distributed cryptocurrency network*, Online, 2018. [Online]. Available: <https://nano.org/en/whitepaper>.
- [94] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and probabilistic leaderless BFT consensus through metastability," *arXiv preprint*, 2019. arXiv: 1906.08936.
- [95] S. Müller, A. Penzkofer, N. Polyanskii, J. Theis, W. Sanders, and H. Moog, "Tangle 2.0: Leaderless nakamoto consensus on the heaviest dag," *IEEE Access*, vol. 10, pp. 105 807–105 842, 2022.
- [96] Q. Wang, J. Yu, S. Chen, and Y. Xiang, *Sok: Diving into DAG-based blockchain systems*, Online, 2022. [Online]. Available: <https://arxiv.org/abs/2012.06128>.
- [97] H. Pervez, M. Muneeb, M. U. Irfan, and I. U. Haq, "A comparative analysis of DAG-based blockchain architectures," in *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, Lahore, Pakistan, 2018, pp. 27–34.
- [98] B. Lashkari and P. Musilek, "A comprehensive review of blockchain consensus mechanisms," *IEEE Access*, vol. 9, pp. 43 620–43 652, 2021.
- [99] T. Sypherd, M. Diaz, L. Sankar, and P. Kairouz, "A tunable loss function for binary classification," in *2019 IEEE International Symposium on Information Theory (ISIT)*, Paris, France, 2019, pp. 2479–2483.

- [100] A. Jadbabaie, P. Molavi, A. Sandroni, and A. Tahbaz-Salehi, “Non-bayesian social learning,” *Games and Economic Behavior*, vol. 76, no. 1, pp. 210–225, 2012.
- [101] S. Shahrapour, A. Rakhlin, and A. Jadbabaie, “Distributed detection: Finite-time analysis and impact of network topology,” *IEEE Transactions on Automatic Control*, vol. 61, no. 11, pp. 3256–3268, Nov. 2016.
- [102] A. Nedić, A. Olshevsky, and C. A. Uribe, “Fast convergence rates for distributed non-bayesian learning,” *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5538–5553, Nov. 2017.
- [103] Q. Liu, A. Fang, L. Wang, and X. Wang, “Social learning with time-varying weights,” *Journal of Systems Science and Complexity*, vol. 27, no. 3, pp. 581–593, 2014.
- [104] L. Su and N. H. Vaidya, “Defending non-bayesian learning against adversarial attacks,” *Distributed Computing*, vol. 32, pp. 277–289, 2019.
- [105] A. Mitra, J. A. Richards, and S. Sundaram, “A new approach to distributed hypothesis testing and non-bayesian learning: Improved learning rate and byzantine resilience,” *IEEE Transactions on Automatic Control*, vol. 66, no. 9, pp. 4084–4100, Sep. 2021.
- [106] A. Hoehn and P. Zhang, “Detection of replay attacks in cyber-physical systems,” in *2016 American Control Conference (ACC)*, Boston, MA, USA, 2016, pp. 290–295.
- [107] R. Han, J. Yu, H. Lin, S. Chen, and P. Veríssimo, *On the security and performance of blockchain sharding*, IACR Cryptology ePrint Archive, 2021.
- [108] S. Feng, J. He, and M. X. Cheng, “Security analysis of block withholding attacks in blockchain,” in *ICC 2021 – IEEE International Conference on Communications*, Montreal, QC, Canada, 2021, pp. 1–6.
- [109] D. Dziubałtowska, “The security of the coordicide: The implementation and analysis of possible attack vectors,” *arXiv*, 2022. arXiv: 2205.12568. [Online]. Available: <https://arxiv.org/abs/2205.12568>.
- [110] P. Perazzo, A. Arena, and G. Dini, “An analysis of routing attacks against iota cryptocurrency,” in *2020 IEEE International Conference on Blockchain (Blockchain)*, Rhodes, Greece, 2020, pp. 517–524.
- [111] I. Foundation, *Tangle 2.0 simulator*, Online, 2021. [Online]. Available: <https://github.com/iotaledger/multiverse-simulation>.
- [112] “Goshimmer docker network tool,” IOTA Foundation. (2023), [Online]. Available: <https://github.com/iotaledger/goshimmer/tree/develop/tools/docker-network>.

- [113] *Goshimmer orphanage*, Online. [Online]. Available: <https://github.com/daria305/goshimmer-orphanage>.
- [114] I. Baumgart, B. Heep, and S. Krause, “Oversim: A flexible overlay network simulation framework,” in *Proc. 10th IEEE Global Internet Symposium (GI '07) in Conjunction with IEEE INFOCOM 2007*, Anchorage, AK, USA, May 2007, pp. 79–84.
- [115] A. Varga and R. Hornig, “An overview of the omnet++ simulation environment,” in *Proc. 1st Int. Conf. Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (SIMUTools)*, 2008, p. 60.
- [116] *Bitcoin core*, Online, 2021. [Online]. Available: <https://bitcoincore.org>.
- [117] L. N. Nguyen, T. D. T. Nguyen, T. N. Dinh, and M. T. Thai, “Optchain: Optimal transactions placement for scalable blockchain sharding,” in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, Dallas, TX, USA, 2019, pp. 525–535.
- [118] *Scikit-learn: Machine learning in python*, Online, 2021. [Online]. Available: <https://scikit-learn.org/stable/>.
- [119] G. Louppe, “Understanding random forests: From theory to practice,” *arXiv preprint*, 2014. arXiv: 1407.7502.
- [120] P. Ferraro, C. King, and R. Shorten, “On the stability of unverified transactions in a DAG-based distributed ledger,” *IEEE Transactions on Automatic Control*, vol. 65, no. 9, pp. 3772–3783, Sep. 2020.
- [121] Z. Liu *et al.*, “Make web3.0 connected,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 2965–2981, Sep. 2022, Sept.–Oct.
- [122] R. Qin *et al.*, “Web3-based decentralized autonomous organizations and operations: Architectures, models, and mechanisms,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 4, pp. 2073–2082, Apr. 2023.
- [123] Z. Liu *et al.*, “Hyperservice: Interoperability and programmability across heterogeneous blockchains,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 549–566.
- [124] W. Liu, B. Cao, M. Peng, and B. Li, “Distributed and parallel blockchain: Towards a multi-chain system with enhanced security,” *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 1, pp. 723–739, Jan. 2025, Jan.–Feb.
- [125] L. Yin, J. Xu, and Q. Tang, “Sidechains with fast cross-chain transfers,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3925–3940, Nov. 2022, Nov.–Dec.

- [126] C. Chen *et al.*, “When digital economy meets web3.0: Applications and challenges,” *IEEE Open Journal of the Computer Society*, vol. 3, pp. 233–245, 2022.
- [127] L. Yin, J. Xu, K. Liang, and Z. Zhang, “Sidechains with optimally succinct proof,” *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 4, pp. 3375–3389, Jul. 2024, July–Aug.
- [128] L. Cao, “Decentralized ai: Edge intelligence and smart blockchain, metaverse, web3, and descii,” *IEEE Intelligent Systems*, vol. 37, no. 3, pp. 6–19, May 2022, May/June.
- [129] A. Taherpour and X. Wang, “A high-throughput and secure coded blockchain for iot,” *IEEE Transactions on Dependable and Secure Computing*, Early Access.
- [130] Y. Chen and C. Bellavitis, “Blockchain disruption and decentralized finance: The rise of decentralized business models,” *Journal of Business Venturing Insights*, vol. 13, e00151, 2020.
- [131] X. Han, Y. Yuan, and F.-Y. Wang, “A blockchain-based framework for central bank digital currency,” in *Proc. IEEE Int. Conf. Service Operations Logistics Inform.*, 2019, pp. 263–268.
- [132] S. A. Abeyratne and R. P. Monfared, “Blockchain ready manufacturing supply chain using distributed ledger,” *International Journal of Research in Engineering and Technology*, vol. 5, no. 9, pp. 1–10, 2016.
- [133] C. C. Agbo, Q. H. Mahmoud, and J. M. Eklund, “Blockchain technology in healthcare: A systematic review,” *Healthcare*, 2019.
- [134] T. McGhin, K. R. Choo, C. Z. Liu, and D. He, “Blockchain in healthcare applications: Research challenges and opportunities,” *Journal of Network and Computer Applications*, vol. 135, pp. 62–75, 2019.
- [135] A. Taherpour and X. Wang, “Hybridchain: Fast, accurate, and secure transaction processing with distributed learning,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 6, pp. 968–982, Jun. 2024.
- [136] P. Gazi, A. Kiayias, and D. Zindros, “Proof-of-stake sidechains,” in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 139–156.
- [137] S. D. Lerner, *Rsk: Bitcoin powered smart contracts*, Online, 2019. [Online]. Available: <https://bit.ly/30PKof>.
- [138] K. Wang, Z. Zhang, and H. S. Kim, “Reviewchain: Smart contract based review system with multi-blockchain gateway,” in *Proc. IEEE Int. Conf. Internet Things, IEEE Green Comput. Commun., IEEE Cyber Phys. Social Comput., IEEE Smart Data*, 2018, pp. 1521–1526.

- [139] A. Xiong, G. Liu, Q. Zhu, A. Jing, and S. W. Loke, “A notary group-based cross-chain mechanism,” *Digital Communications and Networks*, vol. 8, pp. 1059–1067, 2022.
- [140] *Renvm white paper*, Online, 2022. [Online]. Available: <https://github.com/renproject/ren/wiki>.
- [141] E. J. Scheid, T. Hegnauer, B. Rodrigues, and B. Stiller, “Bifröst: A modular blockchain interoperability api,” in *Proc. IEEE 44th Conf. Local Comput. Netw.*, 2019, pp. 332–339.
- [142] T. Nolan, *Alt chains and atomic transfers*, Bitcoin Forum (Online), 2013. [Online]. Available: <https://archive.ph/wWzna>.
- [143] M. Herlihy, “Atomic cross-chain swaps,” in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2018, pp. 245–254.
- [144] B. Pillai, K. Biswas, Z. Hóu, and V. Muthukkumarasamy, “Burn-to-claim: An asset transfer protocol for blockchain interoperability,” *Computer Networks*, vol. 200, p. 108 495, 2021.
- [145] *Btcrelay: Ethereum contract for bitcoin spv*, Online, 2017. [Online]. Available: <https://github.com/ethereum/btcrelay>.
- [146] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. J. Knottenbelt, “Xclaim: Trustless, interoperable, cryptocurrency-backed assets,” in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 193–210.
- [147] M. Westerkamp and M. Diez, “Verilay: A verifiable proof of stake chain relay,” *arXiv*, 2022. arXiv: 2201.08697.
- [148] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, “Tesseract: Real-time cryptocurrency exchange using trusted hardware,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1521–1538.
- [149] *Interledger protocol v4 (ilpv4)*, Online, 2020. [Online]. Available: <https://archive.ph/QZSOR>.
- [150] *The perun framework*, Online, 2022. [Online]. Available: <https://archive.ph/VpMW1>.
- [151] E. Abebe *et al.*, “Enabling enterprise blockchain interoperability with trusted data transfer (industry track),” in *Proc. 20th Int. Middleware Conf. Ind. Track*, 2019, pp. 29–35.
- [152] A. Pupyshv *et al.*, “Gravity: A blockchain-agnostic cross-chain communication and data oracles protocol,” *arXiv*, 2020. arXiv: 2007.00966.

- [153] Z. Ge, D. Loghin, B. C. Ooi, P. Ruan, and T. Wang, “Hybrid blockchain database systems: Design and performance,” *Proceedings of the VLDB Endowment*, vol. 15, no. 5, pp. 1092–1104, 2022.
- [154] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, “Untangling blockchain: A data processing view of blockchain systems,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 2003–2024, Oct. 2018.
- [155] M. J. Amiri, D. Agrawal, and A. E. Abbadi, “Caper: A cross-application permissioned blockchain,” *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1385–1398, 2019.
- [156] V. Ramakrishna, *Meet weaver, one of the new hyperledger labs taking on cross-chain and off-chain operations*, Online, 2021. [Online]. Available: <https://archive.ph/rYjcG>.
- [157] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, “A survey on security and privacy issues of bitcoin,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, Dec. 2018, Fourth Quarter.
- [158] M. Ren *et al.*, “Empirical evaluation of smart contract testing: What is the best choice?” In *Proc. 30th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2021, pp. 566–579.
- [159] M. Kaleem, K. Kasichainula, R. Karanjai, *et al.*, “An event-driven framework for smart contract execution,” in *Proc. 15th ACM Int. Conf. Distrib. Event-based Syst.*, 2021, pp. 78–89.
- [160] B. Bartan and M. Pilanci, “Straggler resilient serverless computing based on polar codes,” in *Proc. 57th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, 2019.
- [161] M. Pilanci, “Computational polarization: An information-theoretic method for resilient computing,” *IEEE Transactions on Information Theory*, vol. 68, no. 4, pp. 2211–2238, Apr. 2022.
- [162] D. Fathollahi and M. Mondelli, “Polar coded computing: The role of the scaling exponent,” in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2022, pp. 2154–2159.
- [163] “Substrate: The blockchain framework for a multichain future,” Parity Technologies. (), [Online]. Available: <https://substrate.io/>.
- [164] *The rust programming language*, Online. [Online]. Available: <https://www.rust-lang.org/>.
- [165] *Libsodium documentation*, Online. [Online]. Available: <https://doc.libsodium.org/>.
- [166] *Ethereum bigquery public dataset for smart contract analytics*, Online. [Online]. Available: <https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-public-dataset-smart-contract-analytics>.

- [167] S. Müller, I. Amigo, A. Reiffers-Masson, and S. Ruano-Rincón, “Stability of local tip pool sizes,” *arXiv*, 2023. [Online]. Available: <https://arxiv.org/abs/2302.01625>.
- [168] S. Hong and J. Chae, “Distributed online learning with multiple kernels,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 3, pp. 1263–1277, Mar. 2023.
- [169] A. P. Kalapaaking, I. Khalil, X. Yi, K.-Y. Lam, G.-B. Huang, and N. Wang, “Auditable and verifiable federated learning based on blockchain-enabled decentralization,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 102–115, Jan. 2025.
- [170] G. Yu *et al.*, “Ironforge: An open, secure, fair, decentralized federated learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 354–368, Jan. 2025.
- [171] H. Kim, J. Park, M. Bennis, and S.-L. Kim, “Blockchained on-device federated learning,” *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, Jun. 2020.
- [172] S. Zhou, H. Huang, W. Chen, P. Zhou, Z. Zheng, and S. Guo, “Pirate: A blockchain-based secure framework of distributed machine learning in 5g networks,” *IEEE Network*, vol. 34, no. 6, pp. 84–91, Dec. 2020, Nov./Dec.
- [173] E. Baccarelli, M. Scarpiniti, A. Momenzadeh, and S. S. Ahrabi, “AfaFed—asynchronous fair adaptive federated learning for iot stream applications,” *Computer Communications*, vol. 195, pp. 376–402, Nov. 2022.
- [174] K. E. Tan, “Achieving security and privacy in federated learning systems: Survey, research challenges and future directions,” *Engineering Applications of Artificial Intelligence*, vol. 106, Nov. 2021.
- [175] X. Cheng, W. Tian, F. Shi, M. Zhao, S. Chen, and H. Wang, “A blockchain-empowered cluster-based federated learning model for blade icing estimation on iot-enabled wind turbine,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 9184–9195, Dec. 2022.
- [176] L. Cui, X. Su, and Y. Zhou, “A fast blockchain-based federated learning framework with compressed communications,” *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 12, pp. 3358–3372, Dec. 2022.
- [177] C. Feng, B. Liu, K. Yu, S. K. Goudos, and S. Wan, “Blockchain empowered decentralized horizontal federated learning for 5g-enabled uavs,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3582–3592, May 2021.
- [178] A. Imteaj, M. H. Amini, and P. M. Pardalos, *Foundations of Blockchain: Theory and Applications*. Cham, Switzerland: Springer, 2021.

- [179] D. C. Nguyen *et al.*, “Federated learning meets blockchain in edge computing: Opportunities and challenges,” *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 806–12 825, Aug. 2021.
- [180] S. Wang *et al.*, “Adaptive federated learning in resource constrained edge computing systems,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.
- [181] Y. Zhao *et al.*, “Privacy-preserving blockchain-based federated learning for iot devices,” *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1817–1829, Feb. 2021.
- [182] Z. Luan, W. Li, M. Liu, and B. Chen, “Robust federated learning: Maximum correntropy aggregation against byzantine attacks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 62–75, Jan. 2025.
- [183] X. Tang, M. Shen, Q. Li, L. Zhu, T. Xue, and Q. Qu, “Pile: Robust privacy-preserving federated learning via verifiable perturbations,” *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 6, pp. 5005–5023, 2023, Nov.–Dec.
- [184] L. Zhu, Z. Liu, and S. Han, “Deep leakage from gradients,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 14 747–14 756.
- [185] X. Luo, Y. Wu, X. Xiao, and B. C. Ooi, “Feature inference attack on model predictions in vertical federated learning,” in *Proc. 2021 IEEE 37th Int. Conf. Data Engineering (ICDE)*, 2021, pp. 181–192.
- [186] M. Lam, G.-Y. Wei, D. Brooks, V. J. Reddi, and M. Mitzenmacher, “Gradient disaggregation: Breaking privacy in federated learning by reconstructing the user participant matrix,” in *Proc. International Conference on Machine Learning (ICML)*, PMLR, 2021, pp. 5959–5968.
- [187] M. Abadi, A. Chu, I. Goodfellow, *et al.*, “Deep learning with differential privacy,” in *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 308–318.
- [188] K. Tam, L. Li, B. Han, C. Xu, and H. Fu, “Federated noisy client learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 1799–1812, Jan. 2025.
- [189] H. Wang, Y. Yu, and Y. Jiang, “Fully decentralized multiagent communication via causal inference,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 12, pp. 10 193–10 202, Dec. 2023.
- [190] T. Liu, X. Xie, and Y. Zhang, “Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy,” in *Proc. 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021, pp. 2968–2985.

- [191] T. Ruckel, J. Sedlmeir, and P. Hofmann, “Fairness, integrity, and privacy in a scalable blockchain-based federated learning system,” *Computer Networks*, vol. 202, p. 108 621, 2022.
- [192] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, *Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences*, Online, 2021.
- [193] F. McKeen *et al.*, “Innovative instructions and software model for isolated execution,” in *Proc. 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013, pp. 10–16.
- [194] R. Poddar, T. Bozic, R. A. Popa, and I. Stoica, “Verifiable oblivious storage,” in *Proc. 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019, pp. 1699–1716.
- [195] J. Groth, “On the size of pairing-based non-interactive arguments,” in *Advances in Cryptology – EUROCRYPT 2016*, ser. Lecture Notes in Computer Science, vol. 9665, Springer, 2016, pp. 305–326.
- [196] B. Parno, C. Gentry, J. Howell, and M. Raykova, “Pinocchio: Nearly practical verifiable computation,” in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 238–252.
- [197] A. D. Santis and G. Persiano, “Zero-knowledge proofs of knowledge without interaction,” in *Proc. 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, Pittsburgh, PA, USA, 1992, pp. 427–436.
- [198] P. Ghosh, “The state-of-the-art in zero-knowledge authentication proof for cloud,” in *Machine Learning Techniques and Analytics for Cloud Security*, Wiley, 2022, pp. 149–170.
- [199] M. Döring, “Simple, strict, proper, and directed: Comparing reachability in directed and undirected temporal graphs,” *arXiv*, Jan. 2025. arXiv: 2501.11697. [Online]. Available: <https://arxiv.org/abs/2501.11697>.
- [200] G. Kritikakis and I. G. Tollis, “Fast and practical dag decomposition with reachability applications,” *arXiv*, Dec. 2022. arXiv: 2212.03945. [Online]. Available: <https://arxiv.org/abs/2212.03945>.
- [201] N. Boria, G. Cabodi, P. Camurati, M. Palena, P. Pasini, and S. Quer, “A greedy approach to answer reachability queries on dags,” *arXiv*, Nov. 2016. arXiv: 1611.02506. [Online]. Available: <https://arxiv.org/abs/1611.02506>.
- [202] Q. Li, B. He, and D. Song, “Model-contrastive federated learning,” in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 10 708–10 717.

- [203] H. Lycklama, L. Burkhalter, A. Viand, N. K uchler, and A. Hithnawi, “Roff: Robustness of secure federated learning,” in *Proc. 2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 453–476.
- [204] H. Song, Y. Wei, Z. Qu, and W. Wang, “Unveiling decentralization: A comprehensive review of technologies, comparison, challenges in bitcoin, ethereum, and solana blockchain,” *arXiv*, Apr. 2024. arXiv: 2404.04841. [Online]. Available: <https://arxiv.org/abs/2404.04841>.
- [205] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.
- [206] R. K. Ghosh and H. Ghosh, “Clock synchronization and event ordering,” in *Distributed Systems: Theory and Applications*, IEEE, 2023, pp. 91–125.
- [207] G. Baruch, M. Baruch, and Y. Goldberg, “A little is enough: Circumventing defenses for distributed learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [208] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020, pp. 2938–2948.
- [209] C. Xie, T. Dinh, A. Mukherjee, and C.-J. Hsieh, “Dba: Distributed backdoor attacks against federated learning,” in *Proc. Int. Conf. Learning Representations (ICLR)*, 2020.
- [210] S. Yuan. “Chainsfl: A blockchain-based federated learning implementation.” (2021), [Online]. Available: <https://github.com/shuoyuan/ChainsFL-implementation>.
- [211] E. Shao. “Blade-fl: Blockchain assisted decentralized federated learning.” (2020), [Online]. Available: <https://github.com/ElvisShaoYumeng/BLADE-FL>.
- [212] “Tensorflow federated: Machine learning on decentralized data,” TensorFlow. (), [Online]. Available: <https://www.tensorflow.org/federated>.
- [213] “Webassembly,” World Wide Web Consortium (W3C). (), [Online]. Available: <https://webassembly.org/>.
- [214] “Grpc: A high-performance, open-source universal rpc framework,” gRPC Authors. (), [Online]. Available: <https://grpc.io/>.
- [215] “Apache kafka: A distributed streaming platform,” Apache Software Foundation. (), [Online]. Available: <https://kafka.apache.org/>.

- [216] “Ezkl: Easy zero-knowledge inference,” ZKonduit. (2023), [Online]. Available: <https://github.com/zkonduit/ezkl>.
- [217] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun. “Zkml: Trustless machine learning for all.” (2023), [Online]. Available: <https://github.com/ddkang/zkml>.
- [218] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4510–4520.
- [219] C. So, *Folding circom circuits: A zkml case study*, Online resource.
- [220] L. Chong, *Zator: Verified inference of a 512-layer neural network using recursive snarks*, Online resource.
- [221] M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” in *Proc. 22nd ACM SIGSAC Conf. Computer and Communications Security (CCS)*, 2015, pp. 1322–1333.
- [222] P. Kairouz, H. B. McMahan, B. Aly, *et al.*, “Advances and open problems in federated learning,” *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
- [223] K. Bonawitz, V. Ivanov, B. Kalia, H. B. McMahan, S. Patel, and D. Ramage, “Practical secure aggregation for privacy-preserving machine learning,” in *Proc. 2017 ACM SIGSAC Conf. Computer and Communications Security (CCS)*, 2017, pp. 1175–1191.
- [224] J. So, K. Lee, E. Kim, and J. Jang, “Gentle: A lightweight secure aggregation protocol for federated learning,” in *Proc. Network and Distributed System Security Symposium (NDSS)*, Paper 19, 2023. [Online]. Available: <https://www.ndss-symposium.org/ndss-paper/gentle-a-lightweight-secure-aggregation-protocol-for-federated-learning/>.
- [225] P. Mohassel, A. C. (Ayush?) And F. Koushanfar, “Secureml: A system for scalable privacy-preserving machine learning,” in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 359–376.
- [226] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in *Proc. 2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 3–18.
- [227] A. Salem, Y. Zhang, M. Javed, M. Stamm, A. Caliskan, and F. Koushanfar, “MI-leaks: Model and data independent membership inference attacks and defenses on machine learning models,” *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2019, no. 3, pp. 133–152, 2019.

- [228] J. Cheon, H. Kim, Y. Kang, H. Lee, and J. Lim, “Clipmi: Membership inference attacks with prediction clipping,” in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2022.
- [229] M. Nasr, R. Shokri, and A. Houmansadr, “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning,” in *Proc. 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019, pp. 739–753.
- [230] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.
- [231] Z. Wan, W. Liu, and H. Cui, “Hibechain: A hierarchical identity-based blockchain system for large-scale iot,” *IEEE Transactions on Big Data*, vol. 11, no. 5, pp. 2244–2263, Oct. 2025.
- [232] P. Tekchandani *et al.*, “Blockchain-enabled secure collaborative model learning using differential privacy for iot-based big data analytics,” *IEEE Transactions on Big Data*, vol. 11, no. 1, pp. 141–156, Feb. 2025.
- [233] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, “Randomized gossip algorithms,” *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, Jun. 2006.
- [234] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, “Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5330–5340.
- [235] X. Lian, W. Zhang, C.-J. Hsieh, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in *Proc. International Conference on Machine Learning (ICML)*, 2018, pp. 3043–3052.
- [236] M. Assran, N. Loizou, N. Ballas, and M. G. Rabbat, “Stochastic gradient push for distributed deep learning,” in *Proc. International Conference on Machine Learning (ICML)*, 2019, pp. 344–353.
- [237] A. Koloskova, S. U. Stich, and M. Jaggi, “Decentralized stochastic optimization and gossip algorithms with compressed communication,” in *Proc. International Conference on Machine Learning (ICML)*, 2019, pp. 3478–3487.
- [238] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 119–129.

- [239] S. Wang, D. Li, J. Geng, Y. Gu, and Y. Cheng, “Impact of network topology on the performance of distributed machine learning: Theoretical analysis and practical factors,” in *Proc. IEEE INFOCOM*, 2019, pp. 1729–1737.
- [240] G. Neglia, G. Calbi, D. Towsley, and G. Vardoyan, “The role of network topology for distributed machine learning,” in *Proc. IEEE INFOCOM*, 2019, pp. 2350–2358.
- [241] D. J. Watts and S. H. Strogatz, “Collective dynamics of ‘small-world’ networks,” *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.
- [242] S. Warnat-Herresthal *et al.*, “Swarm learning for decentralized and confidential clinical machine learning,” *Nature*, vol. 594, pp. 265–270, 2021.
- [243] Y. Qu *et al.*, “Decentralized privacy using blockchain-enabled federated learning in fog computing,” *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5171–5183, Jun. 2020.
- [244] J. Li *et al.*, “Blockchain assisted decentralized federated learning (blade-fl): Performance analysis and resource allocation,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2401–2415, 2022.
- [245] S. Popov, *The tangle*, Whitepaper, Oct. 2018.
- [246] Z. Tang, S. Shi, B. Li, and X. Chu, “Gossipfl: A decentralized federated learning framework with sparsified and adaptive communication,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 909–922, Mar. 2023.
- [247] Q. Chen, Z. Wang, H. Wang, and X. Lin, “Feddual: Pair-wise gossip helps federated learning in large decentralized networks,” *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 335–350, 2023.
- [248] A. Tundo, F. Filippini, F. Regonesi, M. Ciavotta, and M. Savi, “Decentralized edge workload forecasting with gossip learning,” *IEEE Transactions on Network and Service Management*, vol. 22, no. 4, pp. 3016–3031, Aug. 2025.
- [249] H. Wang and Y. Chi, “Communication-efficient federated optimization over semi-decentralized networks,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 11, pp. 147–160, 2025.
- [250] H. Wang *et al.*, “Attack of the tails: Yes, you really can backdoor federated learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [251] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” in *Proc. International Conference on Machine Learning (ICML)*, 2018, pp. 5650–5659.

- [252] K. Pillutla, S. M. Kakade, and Z. Harchaoui, “Robust aggregation for federated learning,” *IEEE Transactions on Signal Processing*, vol. 70, pp. 1142–1154, 2022.
- [253] C. Fung, J. Yoon, and I. Beschastnikh, *Mitigating sybils in federated learning poisoning*, arXiv:1808.04866, 2018. [Online]. Available: <https://arxiv.org/abs/1808.04866>.
- [254] B. Wang, Y. Yao, S. Shan, *et al.*, “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks,” in *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2019, pp. 707–723.
- [255] S. Qiao *et al.*, “Lbfl: A lightweight blockchain-based federated learning framework with proof-of-contribution committee consensus,” *IEEE Transactions on Big Data*, vol. 11, no. 4, pp. 1745–1759, Aug. 2025.
- [256] L. Cui, Y. Li, Y. Zhou, Y. Qu, and J. Liu, “Accelerating blockchain-enabled federated learning with clustered clients,” *IEEE Transactions on Big Data*, vol. 11, no. 5, pp. 2148–2161, Oct. 2025.
- [257] S. Yuan, B. Cao, Y. Sun, Z. Wan, and M. Peng, “Secure and efficient federated learning through layering and sharding blockchain,” *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 3, pp. 3120–3134, Jun. 2024.
- [258] J. Chen, D. Wu, S. Guo, F. Qi, and X. Qiu, “Dag-ensefl: Dag-based asynchronous federated learning with ensemble distillation,” *IEEE Transactions on Big Data*, 2025.
- [259] M. Cao, L. Zhang, and B. Cao, “Toward on-device federated learning: A directed acyclic graph-based blockchain approach,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 4, pp. 2028–2042, Apr. 2023.
- [260] Q. Wang, S. Xu, R. Xu, and B. Ai, “A dag-blockchain-assisted federated learning framework in wireless networks: Learning performance and throughput optimization schemes,” *IEEE Transactions on Vehicular Technology*, vol. 74, no. 3, pp. 5097–5113, Mar. 2025.
- [261] X. Cao, M. Fang, J. Liu, and N. Z. Gong, “Fltrust: Byzantine-robust federated learning via trust bootstrapping,” in *Proc. ISOC Network and Distributed System Security Symposium (NDSS)*, 2021.
- [262] C. Xie, O. Koyejo, and I. Gupta, “Zeno++: Robust fully asynchronous sgd,” in *Proc. 37th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 119, 2020, pp. 10 495–10 503. [Online]. Available: <https://proceedings.mlr.press/v119/xie20c.html>.
- [263] S. Josefsson and I. Liusvaara, *Edwards-curve digital signature algorithm (eddsa)*, RFC 8032, Jan. 2017. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc8032>.

- [264] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, “High-speed high-security signatures,” *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.
- [265] Y. Gao, C. Xu, D. Wang, *et al.*, “Strip: A defence against trojan attacks on deep neural networks,” in *Proc. Annual Computer Security Applications Conference (ACSAC)*, 2019, pp. 113–125.
- [266] B. Tran, J. Li, and A. Madry, *Spectral signatures in backdoor attacks*, arXiv:1811.00636, 2018. [Online]. Available: <https://arxiv.org/abs/1811.00636>.
- [267] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, “Data poisoning attacks against federated learning systems,” in *Computer Security – ESORICS 2020*, ser. Lecture Notes in Computer Science, vol. 12308, Springer, 2020, pp. 480–501.
- [268] B. Iglewicz and D. C. Hoaglin, *How to Detect and Handle Outliers*. Milwaukee, WI: ASQC Quality Press, 1993.
- [269] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, “Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median,” *Journal of Experimental Social Psychology*, vol. 49, no. 4, pp. 764–766, 2013.