Computer Modeling in Engineering & Sciences

Tech Science Press

https://doi.org/10.32604/cmes.2026.080871

ARTICLE

A Computational Modeling Framework for Verifiable Computation Offloading in Resource-Constrained IoT Smart Contract Systems Using Zero-Knowledge and Fuzzy Logic

Hong Min[1,*], Yousef Ibrahim Daradkeh[2], Jung Taek Seo[3,*], Mohd Anjum[4] and Sana Shahab[5]

[1]Department of Computing, Gachon University, Seongnam, Republic of Korea
[2]Department of Computer Engineering and Information, College of Engineering in Wadi Alddawasir, Prince Sattam bin Abdulaziz University, Al-Kharj, Saudi Arabia
[3]Department of Information Security, Gachon University, Seongnam, Republic of Korea
[4]Department of Computer Engineering, Aligarh Muslim University, Aligarh, India
[5]Department of Business Administration, College of Business Administration, Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia

*Corresponding Authors: Hong Min. Email: hmin@gachon.ac.kr; Jung Taek Seo. Email: seojt@gachon.ac.kr

Received: 01 March 2026; Accepted: 15 April 2026; Published: 30 June 2026

ABSTRACT: This study presents a computational modeling framework for efficient and secure computation offloading in Internet of Things (IoT)-enabled smart contract systems. The integration of IoT, edge computing, and blockchain introduces significant challenges, including limited device capacity, high verification cost, and scalability constraints. Existing blockchain verification approaches depend on computationally intensive cryptographic operations that are inefficient for resource-constrained IoT devices, resulting in increased latency, energy consumption, and transaction costs. To address these issues, this study proposes the Zero-Knowledge Fuzzy Logic Offloading and Rollup (Z-FLOR) framework, an adaptive and energy-efficient model designed to enable secure and verifiable computation in IoT-based smart contract systems. The proposed framework integrates three key components. First, a zero-knowledge proof-based verification model using the Groth16 zkSNARK module generates compact and privacy-preserving proofs that enable fast and reliable verification. Second, a Fuzzy Logic–Driven Energy-Aware Offloading module dynamically allocates computational tasks between IoT devices, edge servers, and cloud platforms based on energy availability, network delay, and device reliability. Third, an Optimistic Rollup Verification module aggregates proofs off-chain and submits them in batches to reduce gas costs and enhance scalability. Extensive simulation and experimental evaluation across diverse IoT scenarios demonstrate the effectiveness of the proposed computational framework. Results indicate that Z-FLOR achieves 99.7% verification accuracy and 98.9% proof compression efficiency, while gas cost analysis indicates gas cost reductions in the range of 80%–98%. Z-FLOR additionally achieves a 44.0% reduction in latency, 51.0% savings in gas costs, and 38.0% energy consumption compared to baseline approaches. These findings highlight the capability of the proposed approach to serve as a scalable and energy-efficient modeling solution for secure IoT smart contract execution in decentralized environments.

KEYWORDS: Zero-knowledge proofs; zkSNARK; IoT smart contracts; verifiable computation; blockchain scalability; edge computing; rollup architecture

1 Introduction

Blockchain and IoT technologies enable secure, transparent, and decentralized computation in sensor networks, logistics systems, and edge computing platforms. To enable trusted IoT device operation, smart

contracts—self-executing programs in blockchain networks-provide automation, immutability, and verifiability [1]. However, the limited processor, memory, and energy resources of IoT devices make blockchain-IoT integration difficult. Most IoT nodes use lightweight microcontrollers with limited processing and memory, making them unsuitable for the heavy computational tasks in blockchain operations, such as consensus validation, transaction processing, and cryptographic Proof generation [2]. Ethereum-based blockchain systems demand high levels of computing power and energy use owing to the execution of consensus, smart contract validation, and transaction verification. These features may also add latency, operational cost, and scalability bottlenecks, especially where very large-scale IoT systems have high rates of task offloading [3]. Edge computing designs create an intermediary layer between IoT devices and the blockchain to address these issues. Edge nodes can process, gather, and verify data before sending it to the blockchain, reducing the computational load on IoT devices [4]. IoT devices must verify computations elsewhere, raising trust and verifiability concerns. Zero-Knowledge Proof (ZKP) methods, such as zkSNARKs, enable verifiable computation without disclosing sensitive data [5]. Groth16 provides a small proof size and low verification cost, but the prover must now perform the computationally intensive work, leading to the offloading of frameworks to maximize efficiency, scalability, and verifiable trust in IoT-blockchain ecosystems [6].

This research examines a distinctive technological challenge in achieving verifiable computation in smart contract systems within the Internet of Things (IoT) setting: an asymmetry between cryptographic processing requirements and a device's resource constraints [7]. The first constraint, defined as Computational Resource Asymmetry, describes the difference between the capabilities of an IoT device's hardware and those required for zkSNARK proof generation. The fact that device capabilities and proof-generation requirements are separated by a large gap poses a major limitation to the direct implementation of secure verification on IoT nodes [8]. The other significant obstacle is the scalability bottleneck in blockchains, where the rate at which data is generated by massive IoT sensor networks is high, while the transaction-processing capacity of current blockchains is insufficient [9]. Such an imbalance increases latency and operational costs and reduces system responsiveness, especially in applications that demand high levels of verification [10]. The energy-efficiency requirements also make IoT-based verification systems a bit more complicated, since repeatedly sending wireless verification proofs drains the battery. Regular interaction with time may significantly reduce device lifetimes and operational stability, and energy-conscious communication plans are necessary to implement IoT sustainably [11]. Altogether, these constraints and interrelated considerations establish the precise problem space of this study, balancing computational verifiability, blockchain scalability, and energy sustainability within the cost-constrained, IoT-based development of cost-sustainable smart-contract platforms [12].

Existing blockchain-IoT verifiable computation methods have significant drawbacks. Monolithic zkSNARK implementations are either on-chain or off-chain, which slows verification and doesn't support IoT devices [13]. Malicious nodes can misinterpret static offloading schemes because they lack cryptographic verification. Layer-2 blockchain scaling solutions cannot handle the heterogeneity of IoT networks. No framework combines cryptographic verifiability, intelligent offloading, and blockchain scalability [14]. A unified architectural approach is needed due to high verification costs, real-time operational constraints, energy sustainability considerations, and robust security guarantees. IoT applications are uneconomical at current gas prices; key tasks are delayed; efficient protocols reduce energy usage; and decentralization without sufficient verification raises security concerns [15].

1.1 Motivation of the Study

The fast proliferation of IoT systems requires efficient, secure, and scalable computing in decentralized systems. Nevertheless, IoT devices have limited processing power, memory, and energy, making real-time implementation of blockchain and cryptographic algorithms infeasible. Current solutions address verification, offloading, and scalability independently, resulting in inefficient, fragmented designs. Also, static offloading mechanisms do not adapt to dynamic network conditions, and blockchain operations incur high latency and gas costs. These issues underscore the need to adopt a combined strategy. Thus, the motivation of this study is to create an integrated system that integrates adaptive offloading, efficient cryptographic authentication, and scalable blockchain systems to achieve reliable IoT smart contract execution.

1.2 Research Approach

This study introduces the Z-FLOR framework, which combines zero-knowledge proof generation, intelligent task offloading, and blockchain scalability into a single architecture. The framework uses a zkSNARK module (Groth16) to create efficient, privacy-preserving proofs. An offloading module is a dynamically distributed fuzzy-logic module that assigns tasks based on devices' energy, latency, and reliability parameters. Moreover, an Optimistic Rollup verification module combines off-chain proofs to save on gas and enhance scalability. These elements are aligned with a hybrid interaction layer that enables smooth communication and adaptive decision-making across IoT, edge, and blockchain setups, and provides secure, efficient, and verifiable computation.

1.3 Main Contributions of the Study

1. To propose a novel unified Z-FLOR framework that integrates zkSNARK-based verification, adaptive fuzzy logic offloading, and Optimistic Rollup-based blockchain scaling to address computational asymmetry, energy constraints, and scalability in IoT systems.
2. To develop an efficient Groth16-based zkSNARK mechanism combined with rollup aggregation, enabling compact, privacy-preserving, and low-cost verifiable computation while improving blockchain throughput without compromising security.
3. To introduce an adaptive fuzzy logic-driven offloading model that dynamically optimizes task execution across IoT, edge, and cloud layers, outperforming traditional static and threshold-based approaches in responsiveness and energy efficiency.
4. To design a hybrid interoperable architecture that seamlessly integrates IoT devices, edge nodes, and blockchain networks, ensuring coordinated operation between proof generation, offloading decisions, and scalable verification.
5. To demonstrate, through realistic IoT datasets, that the proposed framework achieves 44.0% latency reduction, 51.0% gas cost savings, 38.0% energy conservation, 99.7% verification accuracy, and 98.9% proof compression efficiency, outperforming existing isolated solutions.

1.4 Structure of the Study

The rest of this paper is structured in the following manner. Section 2 examines the existing literature on blockchain-IoT integration, zero-knowledge verification, and scalable offloading methods. Section 3 illustrates the thread model specification and security analysis theorem. Section 4 presents the suggested Z-FLOR methodology, including cryptographic, offloading, and rollup modules. Section 5 represents results analysis, and Section 6 discusses the results and limitations. Lastly, Section 7 concludes the research and outlines future research directions.

2 Related Works

2.1 Blockchain-Based IoT Frameworks and Resource Optimization

Zhang et al. [16] presented a resource-constrained IoT blockchain-DL architecture. Blockchain mining and DL training consume resources; ZPoL consensus solves this. DL model training saves mining energy and protects model privacy. Quality-aware incentives enable meaningful DL mining by IoT devices. For resource-constrained IoT applications, this ZPoL-based design minimizes communication, compute, and storage costs, according to simulations.

Moore et al. [17] reported that blockchain-based decentralized apps use ZKPs to verify calculations without revealing underlying information. ZKML improves ML deployment trustworthiness and enables privacy-preserving dapps. The paper describes the ZKML approach and how blockchain and smart contracts may verify ML model usage without a trusted authority. Researchers in this burgeoning field can use it to compare ZKML implementation frameworks.

Shamsan Saleh [18] provided that Secure, decentralized AI systems are needed to tackle cyber threats as AI use expands in cybersecurity. Blockchain technology enhances AI security and privacy by enabling decentralized, immutable data storage. This systematic literature review provides a taxonomy and discusses blockchain and decentralized AI in cybersecurity, including obstacles and prospects. It highlights their beneficial interaction, gives real-world applications, and suggests future research. Blockchain-enabled decentralized AI can enhance cybersecurity by improving AI security, privacy, and trust, according to the study.

2.2 Zero-Knowledge Proof-Based Verification Mechanisms

Cai et al. [19] reported that Ciphertext-Policy Attribute-Based Encryption (CP-ABE) has been investigated for mobile computing access control, but its decryption overhead limits its application. The blockchain-enabled framework for outsourced decryption in this study ensures verifiability and an exemption mechanism to protect honest decryption cloud servers against misleading claims. Using zkSNARKs for efficient verification and a challenge-response mechanism to reduce evidence-generation costs, the system encourages fair incentives and blockchain-based decentralized outsourcing. The Ethereum implementation reduces gas usage compared to prior systems, preserving decryption costs while improving the number of attributes.

Koulianos et al. [20] stated that UAVs are being used in numerous areas, requiring secure connectivity. Blockchain technology might solve the problem, but public blockchains may compromise privacy. An advanced solution using zk-SNARKs allows UAVs to authenticate as well as disclose location data without revealing sensitive information. Power and CPU utilization were better in larger drones using Zokrates on a Raspberry Pi in a simulated drone environment. The public blockchain was Ethereum, with Solidity smart contracts tested on the Sepolia testnet. UAV communication security study expands using this method.

Zhang et al. [21] provided that distributed computing data trading systems struggle with entity matching, transaction fairness, and data privacy. It provides a data transaction security system in which smart contracts and ZKPs facilitate the creation of proofs. Elliptic curve cryptography for dual encryption and attribute-proof contracts as well as attribute-atomic-matching smart contracts for fine-grained data property alignments, are lightweight cryptographic solutions. Ethereum tests in industrial IoT show consistent performance, minimal costs, and privacy protection.

Shashidhara et al. [22] showed that Blockchain lacks privacy, security, transparency, and immutability. This study analyzes numerous ZKP technologies as solutions, without focusing on their types or performance. Snarkjs, ZoKrates, and Circom were evaluated for proof size, trusted setup, prover, and verification

speeds, and scalability. The Ethereum ZKP case study is a theoretical analysis in practice. The article recommends research to scale blockchain systems using ZKPs, which improve privacy and security.

Keršič et al. [23] showed that ZKPs, used in blockchain-based dapps, can verify computations without revealing information. This approach has been enhanced for machine learning and is now called Zero-Knowledge Machine Learning (ZKML). It makes ML deployments more reliable and enables dapps to develop privacy-protecting apps. The article investigates the ZKML process and its components, showing how blockchain and smart contracts may prove ML model use without a trusted institution. It also reviews ZKML implementation frameworks to help new researchers get started.

2.3 Task Offloading and Adaptive Resource Management

Alzoubi and Mishra [24] demonstrated that Blockchain (BC) bloat due to data expansion requires complete solutions. The research provides on-chain and off-chain BC bloat management strategies. Structure modifications, pruning, sharding, ephemeral BCs, and zk-SNARKs update on-chain consensus mechanisms and database management. Off-chain storage, historical data storage, and light client techniques aim to parallelize and bundle transactions. These methods optimize scalability and resources but may compromise security, privacy, and data integrity. The report also applies these ideas to resource-constrained settings such as IoT and fog computing, and advises further research to evaluate their efficacy.

Xiang et al. [25] presented that Internet of Medical Things Identity and Access Management needs authentication. Blockchain as an Identity Provider, ZKP authentication, and Single Sign-On standards improve IoMT authentication in this study. The computational burden between blockchain and client devices is balanced, improving RAM and computing power efficiency. The suggested approach secures sensitive medical data and simplifies access to the healthcare IoT ecosystem for authorized users by addressing IoMT authentication weaknesses. Table 1 shows the summary of the related works.

Table 1: Summary of the literature review.

Ref.	Method /Framework	Core Technique	Smart Contract Support	Scalability Mechanism	Energy Efficiency	Key Limitations
[16]	ZPoL-based Blockchain-DL	Deep Learning + Blockchain Mining	Yes	Energy-aware mining optimization	Moderate	Lacks verifiable computation and integrated scalability
[19]	CP-ABE zkSNARK Outsourcing	Attribute-Based Encryption + zkSNARK	Yes	Partial off-chain verification	Low	High decryption overhead; limited IoT adaptability
[24]	Blockchain Bloat Mitigation	Pruning, Sharding, Off-chain Storage	No	On-chain & Off-chain scaling	Moderate	Trade-offs between scalability, security, and privacy

(Continued)

Table 1 (continued)

Ref.	Method /Framework	Core Technique	Smart Contract Support	Scalability Mechanism	Energy Efficiency	Key Limitations
[20]	UAV zkSNARK Authentication	zkSNARK-based authentication	Yes	Limited (application-specific)	Low	Restricted to UAV scenarios; lacks generalization
[25]	Blockchain-based SSO (ZKP)	Zero-Knowledge Authentication	Yes	Lightweight distributed validation	Moderate	Focused only on authentication; no offloading support
[21]	ZKP-based Data Trading	Smart Contracts + ZKP	Yes	Transaction-level optimization	Low	No focus on energy efficiency or IoT constraints
[22]	ZKP Framework Survey	Comparative analysis of ZKP tools	No	Not applicable	Not applicable	No implementation or experimental validation
[23]	Zero-Knowledge ML (ZKML)	ZKP for ML model verification	Yes	Limited scalability	Low	High computational complexity; not real-time suitable
[17]	Blockchain Federated Learning	Secure FL + Blockchain	Yes	Distributed learning optimization	Moderate	Ignores IoT resource constraints and offloading
[18]	Blockchain-based Decentralized AI	Blockchain for AI security	Partial	Conceptual scalability approaches	Not evaluated	Lacks practical validation and implementation details

2.4 Research Gap

The current literature covers topics in blockchain-IoT systems, but does not offer an overall solution. As an example, Refs. [16,17] focus on energy-efficient learning and distributed intelligence but do not provide mechanisms for verifiable computation. Papers such as [19–21] also apply zkSNARKs to ensure security and privacy, though they do not address adaptive task offloading and energy limitations in IoT settings. The methods in [18,24] address scalability and system optimization but do not implement them in

practice alongside cryptographic verification. Likewise, authentication is highlighted in [25], and theoretical understanding is given in [22,23] without complete system implementation. In general, these approaches address verification, offloading, and scalability separately, leaving a gap toward a unified, adaptive, and energy-aware framework.

3 Thread Model Specification and Security Analysis Theorem

Z-FLOR architecture is represented as a distributed system (D, E, C), where $D = \{d_i\}$ represents a set of IoT devices, $E = \{e_j\}$ is a set of edge or cloud offloading nodes, and C is a layer of blockchain verification. Every device $d_i \in D$ produces a computation task $\tau_i = (x_i, w_i)$ where x_i is public input and w_i is a secret data used to produce an off-chain zkSNARK proof π_i, which is verified on-chain by the $Verify(x_i, \pi_i)$ function. Several pieces of evidence are combined in a batch B, the integrity of which is ensured with a Merkle root R to allow its effective validation. The opponent is a probabilistic poly-time (PPT) adversary that may compromise a subset of the devices or offloading nodes, tamper with inputs (x_i, w_i), counterfeit proofs π_i, replay transactions or manipulate batch data $B \to B'$. The opponent is computationally constrained and has no ability to violate zkSNARK soundness or collision-resistant hash functions, and at least one honest validator is assumed. Based on these assumptions, the attack surface has malicious computation, forgery of proofs, replay attacks, and manipulation of rollups, and security properties (correctness, privacy, and integrity) are maintained using zkSNARK verification and Merkle-based batch verification. The chances of successful adversarial attacks are limited to a small value ϵ, denoted by $Pr[\text{Attack}] \leq \epsilon$, such that successful attacks are computationally infeasible by standard cryptography.

Security Analysis Theorem

Theorem 1: *The verification mechanism in the Z-FLOR framework prevents acceptance of invalid computation results generated by malicious entities, assuming zkSNARK soundness and collision-resistant hash properties.*

Proof: The Z-FLOR verification framework consists of the following components:

1. A set of computation tasks $\Gamma = \{\tau_i\}$, where each task $\tau_i = (x_i, w_i)$
2. A set of verification entities $\Upsilon = \{v_k\}$, representing blockchain validators responsible for proof validation.
3. A verification function $Verify(x_i, \pi_i)$, which validates zkSNARK proofs generated for each task.
4. A batch aggregation function B, which groups multiple proofs as: $B = \{\pi_1, \pi_2, \ldots, \pi_n\}$
5. An integrity function represented by a Merkle root R, computed as: $R = MerkleRoot(B)$

An adversary attempts to modify computation data or proofs by generating an invalid proof π_i' or modified batch B', such that:

$$Verify(x_i, \pi_i') = 1$$

or

$$MerkleRoot(B') = R$$
$$Pr[Verify(x_i, \pi_i') = 1] \leq \epsilon_1$$
$$Pr[MerkleRoot(B') = R] \leq \epsilon_2$$
$$Pr[Attack] \leq \epsilon_1 + \epsilon_2$$

Since both probabilities are negligible under standard cryptographic assumptions, invalid computations or modified batches cannot be accepted with non-negligible probability.

Conclusion: Thus, as the probability of accepting invalid proofs or altered batch data is limited by negligible numbers ϵ_1 and ϵ_2, the Z-FLOR system provides secure verification as well as integrity of computations against fraudulent proofs injection and data alteration attacks, hence providing reliable and trustworthy behavior under the conventional cryptographic assumptions.

4 Proposed Methodology for Z-FLOR Framework

The multilayer verifiable Z-FLOR is designed for resource-constrained IoT systems operating under changing network and energy conditions. The workflow begins with IoT task generation $T_i = f\,(S_i\,(t), D_i)$, where Dataset [26] provides device-level traces, including energy states E_i, latency patterns L_i i.e., reliability indicators. The fuzzy inference engine uses these parameters $F_{off}(E_i, L_i, \rho_i)$, which adaptively executes tasks locally, at the edge, or in the cloud. Offloaded tasks are combined into arithmetic circuits and handled using Groth16 proof generation $\pi_i = \text{Prove}(T_i, PK)$ using SNARKJS outputs, constraint files, and the Dataset [27] proving keys. Rollups aggregate proofs mechanism is $R_{agg} = \text{MerkleRoot}(\{\pi_i\})$. This allows bulk submission, reducing on-chain interactions and gas costs under real gas prices $C_{gas}(t) = G_{used} \cdot P_{gas}(t)$ from Dataset [28]. The on-chain verifier verifies batched proofs using optimized Groth16 contracts, yielding verification results $V_{on}(\pi_i)$ with good accuracy and low overhead. The integrated workflow of Z-FLOR enhances Latency, gas-cost efficiency, energy saving, and proof compression in big heterogeneous IoT networks.

The Z-FLOR workflow, shown in Fig. 1, uses heterogeneous inputs and cryptographic components to provide scalable, verifiable, and energy-efficient computation offloading for IoT devices. All external inputs, including Energy, are listed in the leftmost section (E_i), Public input (P_i), Ethereum Gas Price Dataset values, prover activation signals (G_i), Network latency parameters from Dataset-27 (L_i), and trust levels (T_i), together with verified/rejected signals that conclude the verification loop. In Phase 1, the Groth16 zkSNARK Module processes IoT-generated tasks and arithmetic circuits $C(x)$ created with circom, trusted-setup keys, and SNARKJS (Dataset [27]) to generate concise proofs π_i. Phase 2, the Fuzzy Logic Offloading Module, computes an offloading choice using the provided proofs and public inputs. $O_i = f(E_i, L_i, T_i)$ employing membership functions and rule-based fuzzy inference. The module decides whether computations should be performed locally, at the edge, or in the cloud, and generates a rollup-ready transaction. Routed to Phase 3, Optimistic Rollup Verification Module. The research uses Merkle-tree construction to aggregate transactions into batched rollup commitments and submit a single state-root update on-chain to reduce gas consumption based on Dataset [28] gas prices. REST/MQTT communication standards coordinate task routing, proof production, rollup submission, and feedback in Phase 4, the Hybrid Interaction Layer. It displays KPI charts of performance metrics. Phase 5 evaluates and assesses key performance indicators like Latency Reduction Rate (LRR), Gas-Cost Reduction (GCR), Energy Conservation Rate (ECR), and Verification Success Rate (VSR) using realistic sensor behaviors and device-state traces from the IoT-Enabled Smart Grid Dataset (Dataset [26]). Fig. 1 shows how Z-FLOR uses datasets to do verifiable computing with low Latency, decreased gas cost, and great energy efficiency [26–28]. IoT device operations are reviewed, offloaded, validated, aggregated, and analyzed.

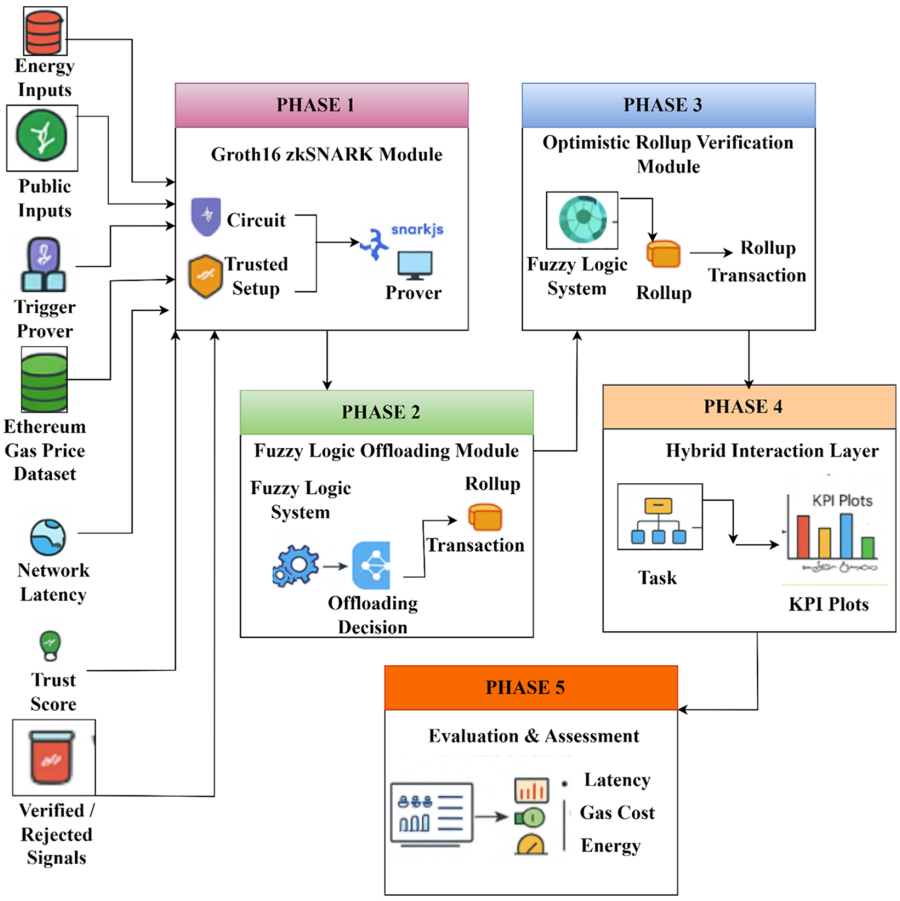


Figure 1: Proposed Z-FLOR system architecture integrating Groth16 zkSNARK module, fuzzy logic offloading module, optimistic rollup verification, hybrid interaction layer, and evaluation pipeline.

4.1 Phase 1: Development of the Groth16 zkSNARK Module

Implementing the Groth16 zkSNARK module in Phase 1 provides the cryptographic foundation for the Z-FLOR framework, enabling privacy-preserving and verifiable offloading of IoT computations. In this module, IoT devices create computations using private inputs (w) and public inputs (x) that are transformed into an arithmetic circuit to represent the logic. The circuit becomes a Quadratic Arithmetic Program (QAP) with polynomial constraint satisfaction. Snarkjs generates the Proving Key (PK) and Verification Key (VK) that tie the circuit structure to the proving system during a trusted setup ritual. The Groth16 prover generates the compact zkSNARK proof by evaluating QAP polynomials and assigning witnesses. $\pi = (\pi_A, \pi_B, \pi_C)$, which is constant in size regardless of circuit complexity. The Groth16 Verifier smart contract verifies this Proof and public inputs x on-chain using pairing-based cryptographic tests. The verifier checks the given evidence for encoded restrictions and returns VALID or INVALID, protecting offloaded computations. This module bridges IoT execution with blockchain-level trust by supporting secure, low-cost, and scalable verifiable computation in the Z-FLOR architecture.

$$
\begin{aligned}
C(w,x) \quad & \longrightarrow \text{Arithmetic Circuit} \\
& \longrightarrow \text{R1CS: } \langle A_i, B_i, C_i \rangle \\
& \longrightarrow \text{QAP Polynomials} \\
& \Rightarrow \begin{cases} F(t) = \sum_i w_i A_i(t) \\ G(t) = \sum_i w_i B_i(t) \\ H(t) = \sum_i w_i C_i(t) \end{cases} \\
& \text{Constraint: } F(t) \cdot G(t) - H(t) = Z(t)
\end{aligned} \tag{1}
$$

$$
\left.
\begin{aligned}
\mathcal{W} &= (w_1, w_2, \ldots, w_n, x_1, x_2, \ldots, x_m) \\
\mathrm{v}(t) &= \textstyle\sum_{i=1}^{n+m} \mathcal{W}_i \cdot v_i(t) \\
\mathrm{u}(t) &= \textstyle\sum_{i=1}^{n+m} \mathcal{W}_i \cdot u_i(t) \\
\mathrm{y}(t) &= \textstyle\sum_{i=1}^{n+m} \mathcal{W}_i \cdot y_i(t)
\end{aligned}
\right\} \tag{2}
$$

$$
\left.
\begin{aligned}
\pi_A &= \alpha + r \cdot \delta_A \\
\pi_B &= \beta + s \cdot \delta_B \\
\pi_C &= H(t) \cdot \delta_C + r \cdot s \cdot \gamma \\
\pi &= (\pi_A, \pi_B, \pi_C)
\end{aligned}
\right\} \tag{3}
$$

To start Phase 1, translate the computation $C(w,x)$ into a verifiable constraint system. Eq. (1) shows how an arithmetic circuit is turned into a Rank-1 Constraint System for IoT computation. Every constraint has a basis polynomial. $A_i(t)$, $B_i(t)$ and $C_i(t)$. The QAP polynomials $F(t), G(t),$ and $H(t)$ are formed by aggregating these terms. The zkSNARK's algebraic foundation relies on the equality $F(t) \cdot G(t) - H(t) = Z(t)$, ensuring computation validity only when all constraints are met. The witness polynomial encodes all private and public inputs based on Eq. (2). The full witness vector W includes private values w_i while the publicly known inputs are denoted by x_i. Values are embedded into polynomial structures using basis functions $v_i(t), u_i(t)$ and $y_i(t)$. The prover proves computation correctness without revealing private values using these polynomials. Eq. (3) summarises the Groth16 prover's internal operations, where randomness r and s contribute to zero-knowledge features. The prover creates three proof elements π_A, π_B and π_C which combine QAP polynomial assessments with trustworthy setup settings. The final zkSNARK proof π is constant-sized and circuit complexity-independent, enabling efficient storage and transmission in resource-limited IoT settings. Finally, Groth16 Verifier's on-chain pairing check. Bilinear proof-verification-key pairings are evaluated by the verifier. The Proof is VALID if all pairings are equal, ensuring that the computation result meets all QAP constraints. Invalid means the equation failed, avoiding fraudulent or inaccurate offloaded computations.

Fig. 2 shows how Quadratic Arithmetic Programs (QAP) and the Groth16 proving system transform an IoT device's calculation into a verifiable ZKP in Phase 1: Development of the Groth16 zkSNARK Module. Starting with the IoT device, computation inputs include private (w) and public (x) inputs. The Circuit Creator turns the calculation $C(w,x)$ into an arithmetic circuit and converts it to a QAP representation $Q = \{F, G, H\}$, allowing constraint satisfaction to be stated as polynomial equations. A trusted setup process using snarkjs generates the Proving Key (PK) and Verification Key when the circuit is constructed. The Groth16 Prover receives the PK, witness assignment w, and public inputs x. The prover then computes the witness, evaluates the constraint polynomials, and uses these evaluations to construct the compact Groth16 proof elements π_A, π_B, π_C, collectively forming the final Proof π. This Proof is concise, constant-sized, and circuit complexity-independent, making it ideal for resource-constrained IoT applications. The resulting proof π is paired with public inputs x and the verification key VK in the Verification Phase and submitted to the Groth16 Verifier smart contract. The verifier checks the given Proof for VK QAP constraints using

pairing-based cryptography. If the pairing check passes, the verifier returns VALID; otherwise, INVALID, ensuring strong integrity guarantees for all offloaded computations. Fig. 2 shows how Groth16 zkSNARKs provide lightweight, secure verifiable computation in the Z-FLOR framework, from circuit conversion and trusted setup to witness computation, proof generation, and on-chain verification.

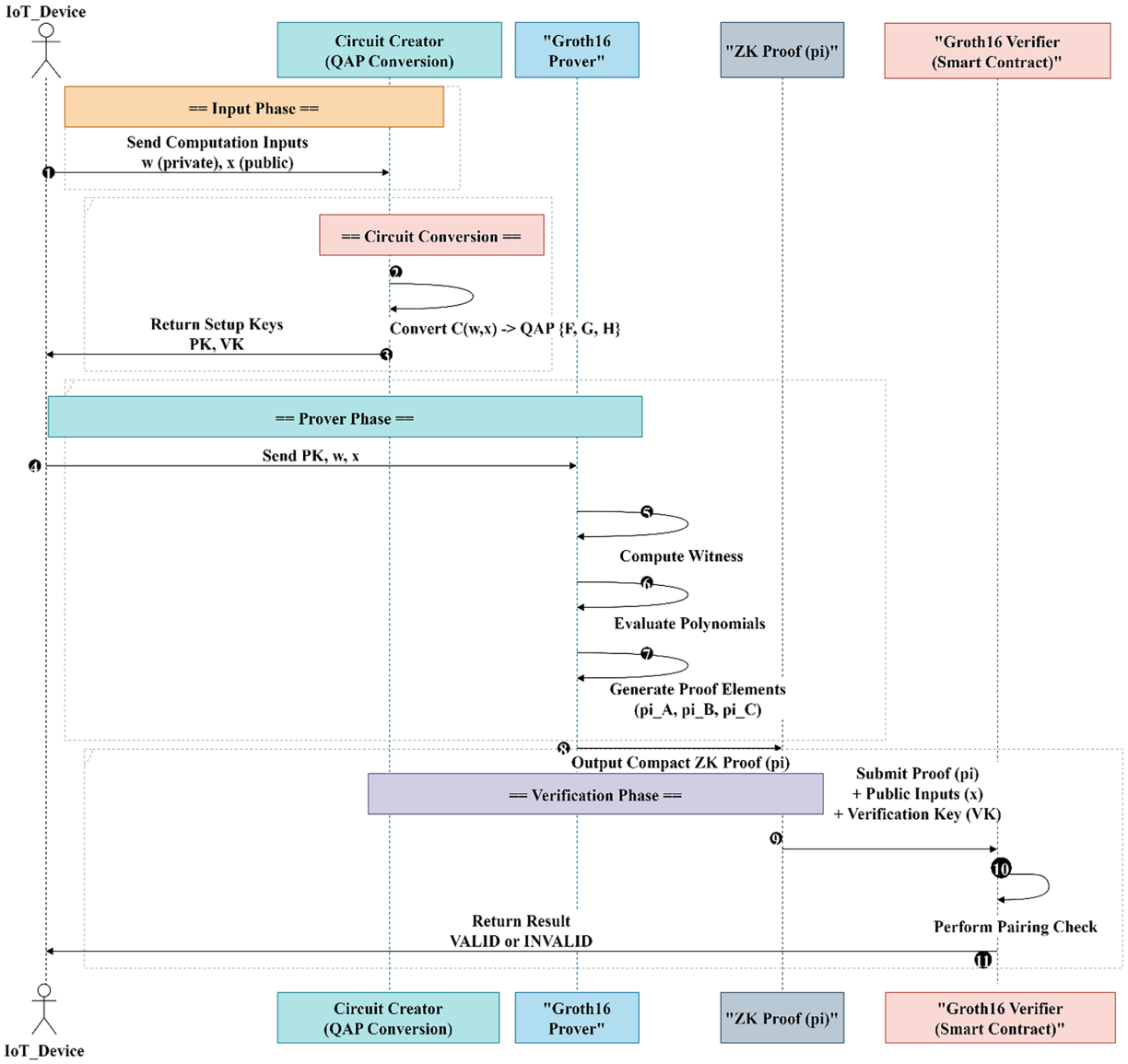


Figure 2: Workflow diagram of phase 1: development of the Groth16 zkSNARK module showing QAP conversion, witness generation, proof construction, and on-chain verification.

Smart Contract Design and Verification Mechanism

The Z-FLOR framework uses a combination of synchronized smart contracts to provide a secure, scalable, and verifiable computation in IoT-blockchain systems. The major contracts are: Groth16 Verifier Contract, Optimistic Rollup Verifier Contract, and Dispute Resolution Contract. All contracts are designed with modular responsibility to enable effective interaction between off-chain and on-chain computation.

Design of Groth16 Verifier Smart Contract: The Groth16 Verifier contract validates zkSNARK proofs generated off-chain. It is coded in Solidity and has a central verification procedure, namely the "Perform Pairing Check," which carries out the cryptographic validation. The elements of proof (A, B, C), the inputs of the parties in the form of x, and the verification key vk are accepted in the contract. The Perform Pairing Check function evaluates bilinear pairing equations over elliptic curve groups, defined as: $e(A, B) = e(\alpha, \beta) \cdot e(\gamma, vk_x) \cdot e(C, \delta)$, where $e(\cdot, \cdot)$ denotes a bilinear pairing function over elliptic curve groups. The proof elements A, B, C are generated by the prover, where $A, C \in G_1$ and $B \in G_2$, representing encoded computation validity. The parameters $\alpha, \beta, \gamma, \delta$ are components of the verification key derived during the trusted setup phase. This internal functionality uses precompiled pairing functions provided in the blockchain setting (e.g., the Ethereum BN128 pairing precompile) to enable efficient execution. If the equality condition is satisfied, the function is true (VALID), assuring that the proof is full of circuit constraints; otherwise, the function is false (INVALID). The design has constant-time verification complexity, O(1), and is thus useful in resource-constrained and cost-sensitive blockchain settings.

Integration with Rollup and Transaction Processing: The Optimistic Rollup Verifier Contract compiles multiple zkSNARK proofs into a single batch transaction, reducing the number of on-chain verification calls. The system checks the root of a compressed state of aggregated computations instead of executing the Perform Pairing Check on each individual proof. The Dispute Resolution Contract tracks this process within the challenge window and, if fraud proofs are submitted, triggers the Groth16 Verifier to perform a Pairing Check, thereby maintaining correctness and accountability.

Relation to AdapT Verification Mechanism Pattern: The architecture of the smart contract in Z-FLOR aligns with the AdapT verification mechanism pattern [29], enabling modular, reusable transaction validation for congruent transaction types. The zkSNARK proof submissions are in a fixed format and are verified by a dedicated verification module. The Perform Pairing Check function is a reusable validation element, much like the validation units defined in AdapT. In the meantime, the Rollup Verifier and Dispute Resolution contracts handle transaction aggregation and exception processing. This isolation of responsibility leads to scalability, minimized redundant calculations, and high maintainability. Z-FLOR uses AdapT principles to ensure scalability and the efficient management of large volumes of proof-based transactions.

Fig. 3 illustrates the Groth16 zkSNARK module workflow, showing the transformation of private and public inputs into an arithmetic circuit, followed by trusted setup and off-chain proof generation. The generated proof is submitted to the blockchain, where the smart contract performs pairing checks to verify its correctness, efficiently producing a valid or invalid result.

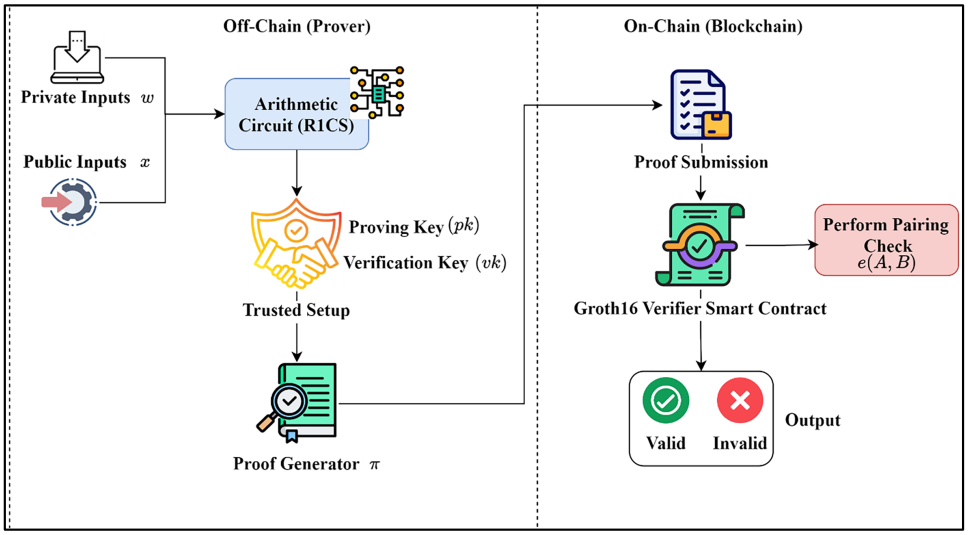


Figure 3: Structure of the Groth 16 zkSNARK module.

4.2 Phase 2: Design of the Fuzzy Logic–Driven Energy-Aware Offloading Module

The Z-FLOR framework's adaptive decision-making core, the Fuzzy Logic–Driven Energy-Aware Offloading Module, debuts in Phase 2. Based on three dynamic device-state characteristics, the module decides whether to conduct an IoT job locally or offload it to edge or cloud resources. The Fuzzy Inference Engine (FIE) at the Off-Chain Processing Layer processes crisp inputs from real-time telemetry in the IoT-Enabled Smart Grid Dataset (Dataset-26). The Fuzzification Unit turns inputs into Low, Medium, and High linguistic categories, while the Knowledge Base applies Energy-, Latency-, and trust-aware rules. After synthesizing these rules, the Decision-Making Unit generates a fuzzy offloading preference, which is then defuzzified into an Offloading Score (O). Local or Offloaded Execution is determined by this score, providing adaptive and energy-efficient task allocation across heterogeneous IoT contexts.

$$\left.\begin{array}{lll} \mu_{E_i}^{\text{Low}}(E_i), & \mu_{E_i}^{\text{Med}}(E_i), & \mu_{E_i}^{\text{High}}(E_i) \\ \mu_{L_i}^{\text{Low}}(L_i), & \mu_{L_i}^{\text{Med}}(L_i), & \mu_{L_i}^{\text{High}}(L_i) \\ \mu_{T_i}^{\text{Low}}(T_i), & \mu_{T_i}^{\text{Med}}(T_i), & \mu_{T_i}^{\text{High}}(T_i) \end{array}\right\} \tag{4}$$

$$\left.\begin{array}{ll} \mu_{\text{Offload}}(y) & = \max_k[\mu_{R_k}(y)], \\ \mu_{\text{Local}}(y) & = \max_m[\mu_{R_m}(y)] \end{array}\right\} \tag{5}$$

$$O_i = \frac{\int_{y_{\min}}^{y_{\max}} y \cdot \mu_{\text{Decision}}(y)dy}{\int_{y_{\min}}^{y_{\max}} \mu_{\text{Decision}}(y)dy} \tag{6}$$

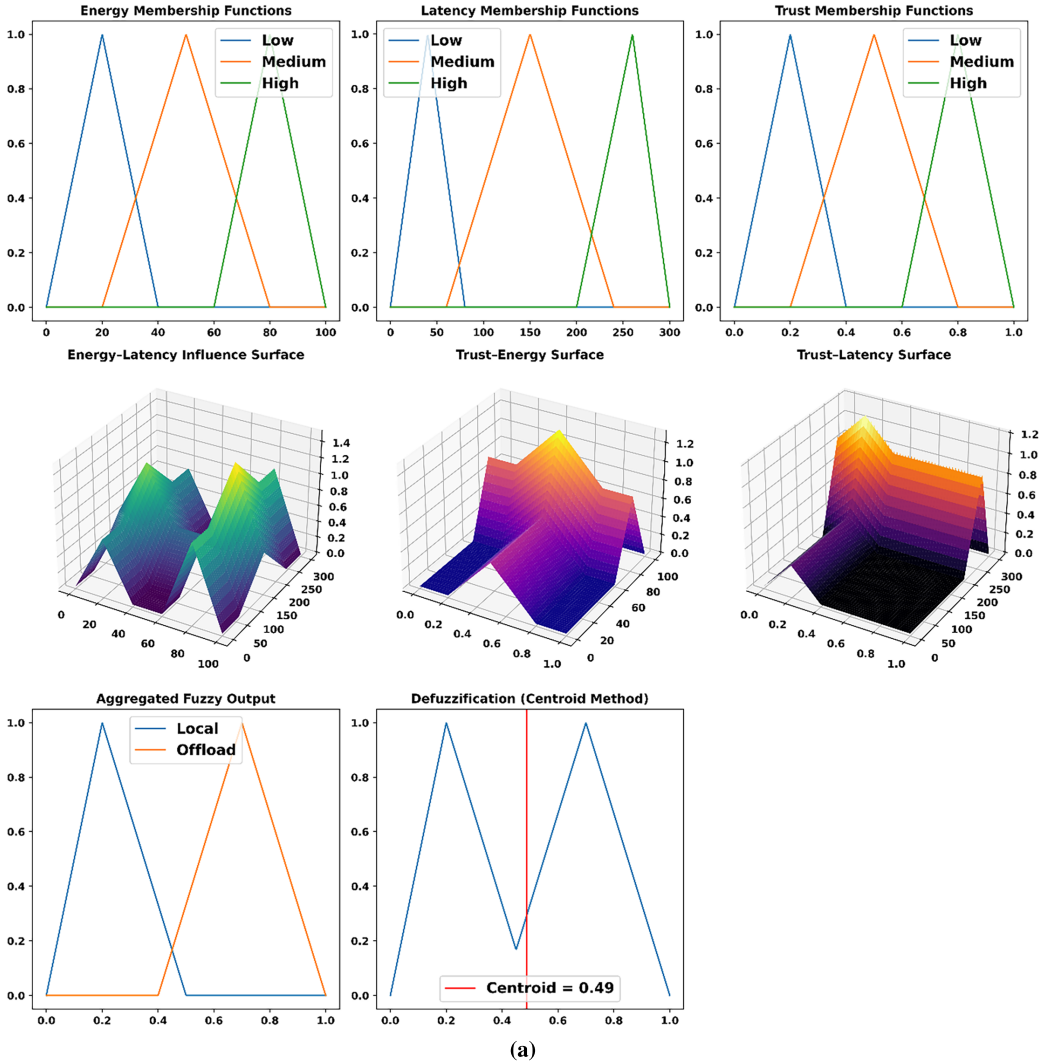
$$\text{Decision} = \begin{cases} \text{Local Execution}, & O_i <; \theta \\ \text{Offloaded Execution}, & O_i \geq \theta \end{cases} \tag{7}$$

The Fuzzy Logic-Based Energy-Aware Offloading Module provides a fuzzy system that uses multi-stage decision-making. The first step involves fuzzification, which transforms the crisp inputs of the devices, i.e., energy level E_i, latency L_i, and trust level T_i, into fuzzy membership degrees as expressed in the equation below: The input parameters are normalized into the range [0, 1], and each is triangularly assigned to three

linguistic variables: Low (L), Medium (M), and High (H) with the help of triangular membership functions. As an example, the energy Ei can be expressed as Low: (0, 0, 0.4), Medium: (0.2, 0.5, 0.8), and High: (0.6, 1, 1), and corresponding mappings can be defined with latency L_i and trust T_i to reflect system uncertainty in terms of the dataset [26].

After fuzzification, the application of a set of expert-established IF-THEN rules can be used to infer preferences of offloading as shown in Fig. 4. Such rules mix fuzzy inputs to capture operational trade-offs, e.g., low energy or high latency states make offloading more likely, whereas high trust and energy states prefer local execution. The rule base of 27 rules (33 combinations) is complete and covers all potential input states. The fuzzy rule base consists of 27 IF–THEN rules covering all possible combinations of input states. For example:

- IF (Energy is Low) AND (Latency is High) AND (Trust is Low) THEN Offload = Cloud;
- IF (Energy is Medium) AND (Latency is Medium) AND (Trust is Medium) THEN Offload = Edge;
- IF (Energy is High) AND (Latency is Low) AND (Trust is High) THEN Offload = Local.



(a)

Figure 4: (Continued)

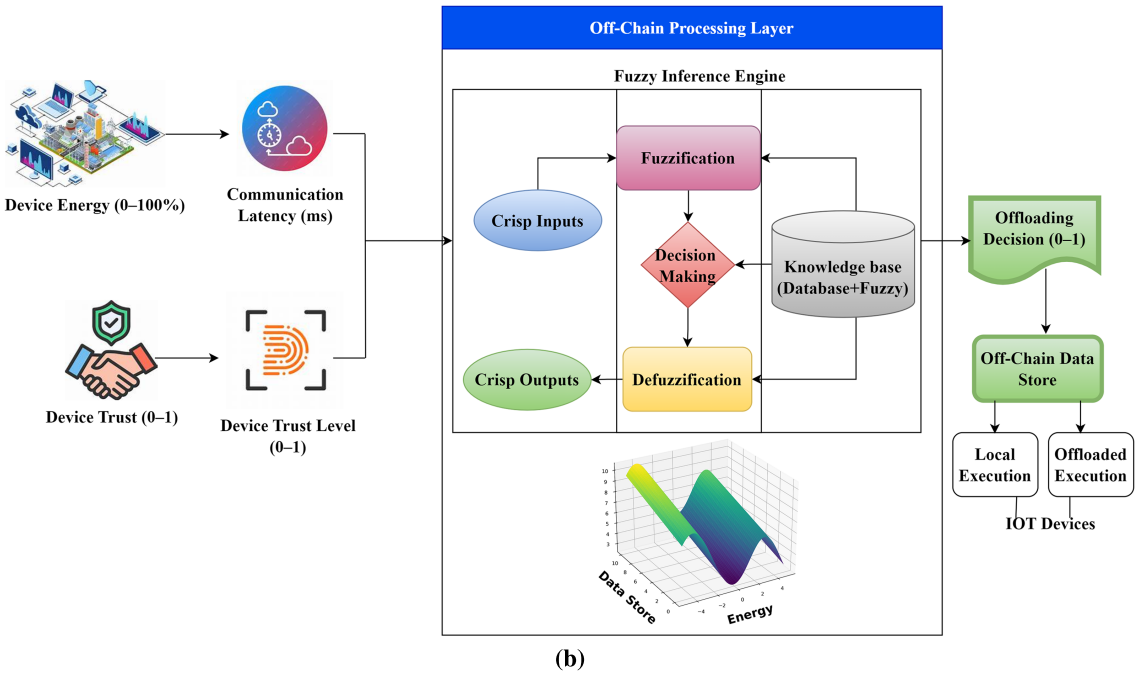


Figure 4: (**a**) Fuzzy membership functions, 3D interaction surfaces, and defuzzification output for the energy-aware offloading module. (**b**) Block-level architecture of the fuzzy Logic–driven energy-aware offloading module, illustrating the integration of device-state inputs, fuzzy inference processing, and decision synthesis for adaptive IoT task offloading.

In Eq. (5), composite fuzzy sets of the probability of Offload and Local Execution decisions are created by adding the output of all the rules that are turned on. This aggregation process combines the contributions of various rules, which essentially address multi-criteria uncertainty in IoT settings. Defuzzification is performed using the centroid method, as defined in Eqs. (6) and (7), to convert the aggregated fuzzy output into a crisp offloading score O_i. This score is the suitability of offloading in general to a task. The final decision is made by comparing O_i with a predetermined threshold θ: when O_i exceeds θ, the task is to be offloaded; when O_i falls below θ, it is to be executed locally. The sensitivity analysis is used to select $\theta = 0.5$ as the threshold value that balances latency and energy consumption. Lower values tilt the system toward offloading and may increase communication delays, whereas higher values tilt the system toward local processing and may increase energy consumption. Overall, these formulations (Eqs. (4)–(7)) provide an adaptive, interpretable, and energy-efficient decision-making mechanism suitable for dynamic IoT scenarios.

The Z-FLOR framework's Phase 2 fuzzy-processing workflow transforms device-state parameters from the IoT-Enabled Smart Grid Dataset (Dataset-26) into an adaptive offloading decision, as shown in Fig. 4a. The membership functions for Energy E_i, Latency L_i, and Trust T_i. Determine fuzzification in Eq. (4). The 3D interaction surfaces visualize the nonlinear rule behaviour formalized in Eq. (5). The aggregated decision sets for Local and Offload, generated using Eq. (6), lead to the centroid-based Offloading Score computed via Eq. (7). This graphic depicts the whole fuzzy workflow, providing interpretable and energy-efficient IoT job allocation.

Fig. 4b shows the internal architecture and operational workflow of the Z-FLOR framework's Phase 2 Fuzzy Logic–Driven Energy-Aware Offloading Module. The Device Energy is computed as $E_i \in [0, 100]\%$ and the module gets three fundamental device-state indicators, Communication Latency L_i, Device Trust Level $T_i \in [0,1]]$ derived from real-time telemetry and dataset-driven simulations. The Fuzzy Inference Engine (FIE) in the off-chain processing layer receives these crisp inputs. The Fuzzification Unit first

converts numerical variables to language terms using established membership functions. The Knowledge Base evaluates fuzzy variables using membership-function definitions and expert-defined rules. Based on multiple criteria such as low-energy, network congestion, and device trustworthiness, the Decision-Making Unit infers a fuzzy-level offloading preference using the rule set. The fuzzy output is processed by the Defuzzification Unit, resulting in a clear offloading score O_i that affects local or offloaded execution preference. The Off-Chain Data Store stores the final choice for downstream integration with Groth16 ZKP and Optimistic Rollup Verification Modules. The figure shows how Energy, Data Size, and Latency affect offloading choice using a three-dimensional response surface. Fuzzy reasoning allows adaptive, interpretable, and resource-efficient task allocation in dynamic IoT situations, improving dependability and energy sustainability.

4.3 Phase 3: Implementation of the Optimistic Rollup Verification Module

Phase 3 utilizes the Optimistic Rollup Verification Module to aggregate Phase 1 zkSNARK proofs for scalable, low-cost validation of offloaded computations. The Off-Chain Processing Layer verifies proofs π_i and public inputs x_i are processed by the Optimistic Rollup Sequencer to create a Merkle commitment. $R = \text{MerkleRoot}(\pi_1, \pi_2, \ldots, \pi_n)$, and calculates the updated rollup state. The new value is the sum of the former values (old and new) $S_{\text{new}} = f(S_{\text{old}}, R)$. The rollup bundle $B = \{R, S_{\text{new}}, \text{TxBatch}\}$ is submitted to the On-Chain Rollup Verifier Contract, where a challenge window Δt is opened. Participants can submit a Fraud Proof identifying index j with the Merkle path during this period M_j and Proof π_j. Dispute Resolution Contract confirms accuracy using $\text{Verify}(\pi_j, x_j, VK)$. Failed verification results in the state being finalized. This module ensures efficient, trust-preserving verification in the Z-FLOR framework.

Fig. 5 illustrates the complete operational workflow of Phase 3: Implementation of the Optimistic Rollup Verification Module, which aggregates multiple zkSNARK proofs generated in Phase 1 and submits a single optimized state update to the blockchain. The process begins in the Off-Chain Processing Layer, where verified proofs π_i and their corresponding public inputs x_i are collected from the Verified & Prover Module. These individual Proof–transaction pairs $\{(\pi_i, x_i)\}_{i=1}^{n}$ are consumed by the Optimistic Rollup Sequencer, which performs Proof Aggregation to construct a Merkle commitment $R = \text{MerkleRoot}(\pi_1, \pi_2, \ldots, \pi_n)$. The sequencer computes a new state root $S_{\text{new}} = f(S_{\text{old}}, R)$, where $f(\cdot)$ represents the deterministic state-transition function derived from the batched transactions.

The Rollup Bundle $\mathcal{B} = \{R, S_{\text{new}}, \text{TxBatch}\}$ is submitted to the On-Chain Rollup Verifier Contract. The blockchain stores the proposed state root S_{new} and activates a Challenge Window of duration Δt upon reception. During this time, participants can file Fraud Proofs to contest batch validity. A challenger discovers a suspicious index j and gives a Merkle path M_j and proof element π_j. The Verification Game in the Dispute Resolution Contract involves reconstructing the proof context and using the Groth16 equation $\text{Verify}(\pi_j, x_j, VK) \in \{\text{TRUE}, \text{FALSE}\}$, where VK is the verification key generated in Phase 1. $Verify = FALSE$ rejects the state root, executes State Reversion, and penalizes the sequencer P_{seq}. The invalid batch is committed to Off-Chain Data Store B, reverting the chain to $S_{\text{final}} = S_{\text{old}}$. If no legitimate challenge is raised within Δt or all fraud proofs fail, the state root is finalized, with $S_{\text{final}} = S_{\text{new}}$. The Verified State & Transaction Log records the validated state, assuring downstream module compatibility. Z-FLOR uses optimistic batching of zkSNARK-based computations to accomplish scalable verification, decreased gas cost, and cryptographic integrity.

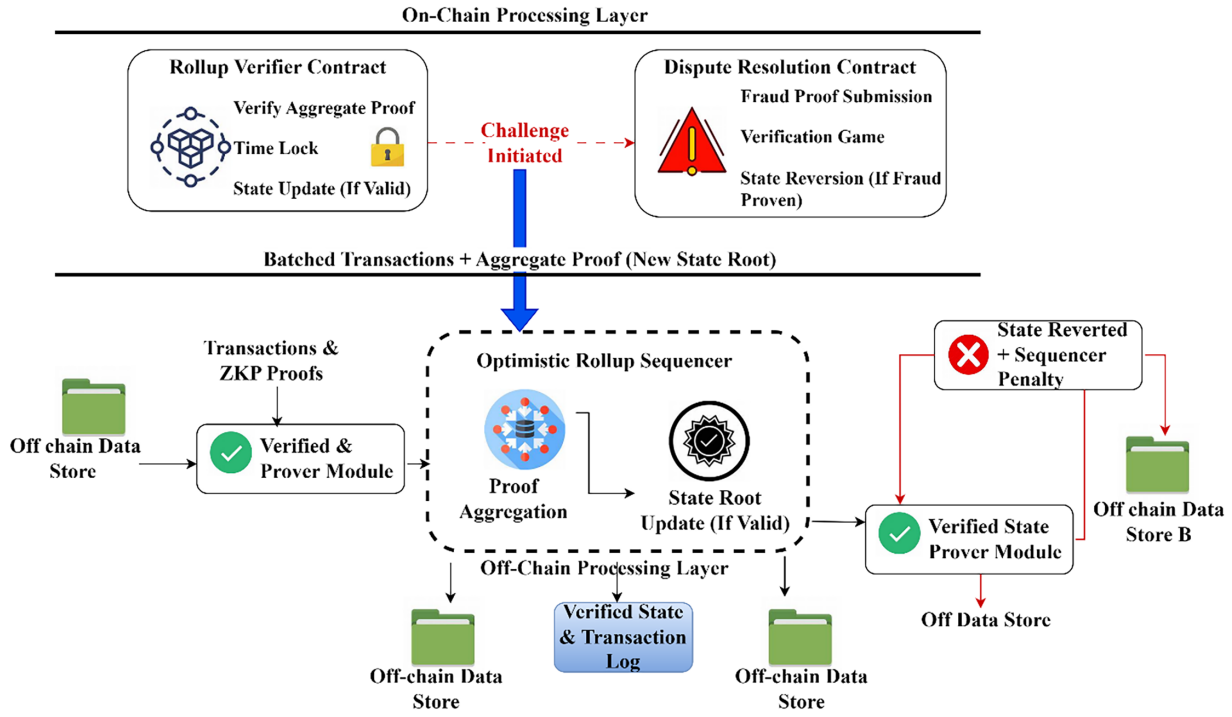


Figure 5: Architectural workflow of the optimistic rollup verification module, illustrating off-chain proof aggregation, state-root submission, challenge handling, fraud-proof verification, and finalized or reverted on-chain state outcomes.

Algorithm 1 illustrates how the Optimistic Rollup Verification Module aggregates Phase 1 zkSNARK proofs to validate offloaded computations scalablely. The process starts with collecting confirmed Proof-input pairs $\{\pi_i, x_i\}$, from IoT computation tasks from Dataset-26. A Merkle commitment $R = \text{MerkleRoot}(\pi_1, \pi_2, \ldots, \pi_n)$, ensures integrity and traceability of all batched computations by aggregating these proofs. After computing the batch effect, the sequencer calculates the updated rollup state $S_{\text{new}} = f(S_{\text{old}}, R)$. Submitting a rollup bundle $B = \{R, S_{\text{new}}, TxBatch\}$ to the on-chain Rollup Verifier Contract creates a challenge window of duration Δt. During this time, participants can submit fraud proofs by specifying an index j , Merkle path M_j, and proof element π_j. The Dispute Resolution Contract uses Groth16's Verify check to verify accuracy $\text{Verify}(\pi_j, x_j, VK)$. The system rejects the batch, reverts to S_{old}, and applies P_{seq} if verification fails. Without a valid challenge before Δt ends, the changed state is placed in the Verified State Log as $S_{\text{final}} = S_{\text{new}}$. This approach preserves efficiency, security, and trust in Z-FLOR rollup-based verification.

Algorithm 1: Optimistic rollup verification module

Input: $\{\pi_i\}, \{x_i\}, S_{old}, VK, \Delta t$
Output: Finalized state S_{final}
1: **procedure** OptimisticRollupVerify($\{\pi_i, x_i\}, S_{old}, VK, \Delta t$)
2: # Aggregate zkSNARK proofs
3: $R = MerkleRoot(\pi^1, \pi^2, \ldots, \pi_n)$
4: # Compute updated rollup state
5: $S_{new} = f(S_{old}, R)$
6: # Create rollup bundle

(Continued)

Algorithm 1 (continued)

$B = \{R, S_{new}, TxBatch\}$
# Submit bundle on-chain
submitToChain(B); activateChallengeWindow(Δt)
# Monitor fraud proofs
while windowOpen():
    if fraudProof(j, M_j, π_j):
        result = Verify(π_j, x_j, VK)
        if result == FALSE:
            $S_{final} = S_{old}; applyPenalty(P_{seq})$
        return S_{final}
# No fraud detected → finalize state
closeWindow()
$S_{final} = S_{new}$
storeFinalState(S_{final}); updateTxLog(TxBatch)
return S_{final}
end procedure

The Z-FLOR framework uses the Optimistic Rollup check system, in which transactions are considered valid unless counterfeited within a challenge period Δt. Validators can also provide evidence of fraud during this time to identify wrong state transitions. The malicious sequencers are penalized by a penalty P_{seq}, which is proportional to the size of the batch, where honest validators are rewarded by an incentive function R_v. In case no challenge is performed during Δt, the batch is closed; in case invalid transactions are rolled back and checked with the help of the pairing check based on Groth16, which guarantees that the system is correct and economically safe.

4.4 Phase 4: Creation of the Hybrid Interaction Layer

Phase 4 creates Z-FLOR's Hybrid Interaction Layer, which synchronizes IoT tasks, offloading options, and zkSNARK verification. Structure and process IoT telemetry and tasks. Subsystem communication, adaptive Decision-Making execution pathway selection, and module data sharing are managed by the layer. It checks the FuzzyOffloadingScheduler for offloading, passes jobs to Groth16Prover, and provides proofs to Optimistic Rollup. Verifying and reporting discoveries on-chain provides safe and smooth computation offloading.

Fig. 6 illustrates the Z-FLOR framework's Hybrid Interaction Layer workflow, integrating IoT device telemetry, fuzzy offloading logic, Groth16 proofs, optimistic rollup aggregation, and dispute-resolution mechanisms for secure and adaptive computation offloading. Dataset [26]-based IoT devices generate task inputs and state metrics that the Computation Tasks Data module processes and routes to the Hybrid Interaction Layer. The layer queries the Fuzzy Offloading Scheduler utilizing parameters (E_i, L_i, T_i) to decide whether to execute locally or offload. The Groth16 Prover computes witnesses and generates zkSNARK proofs (π_i, x_i) for offloaded tasks. Proofs are held off-chain and sent to the Optimistic Rollup module for batching and state-root submission. The Dispute Resolution Contract handles feedback and fraud-proof challenges, validating or updating state transitions with the prover. The workflow guarantees reliable, cost-effective, and trustworthy offloading in diverse IoT contexts.

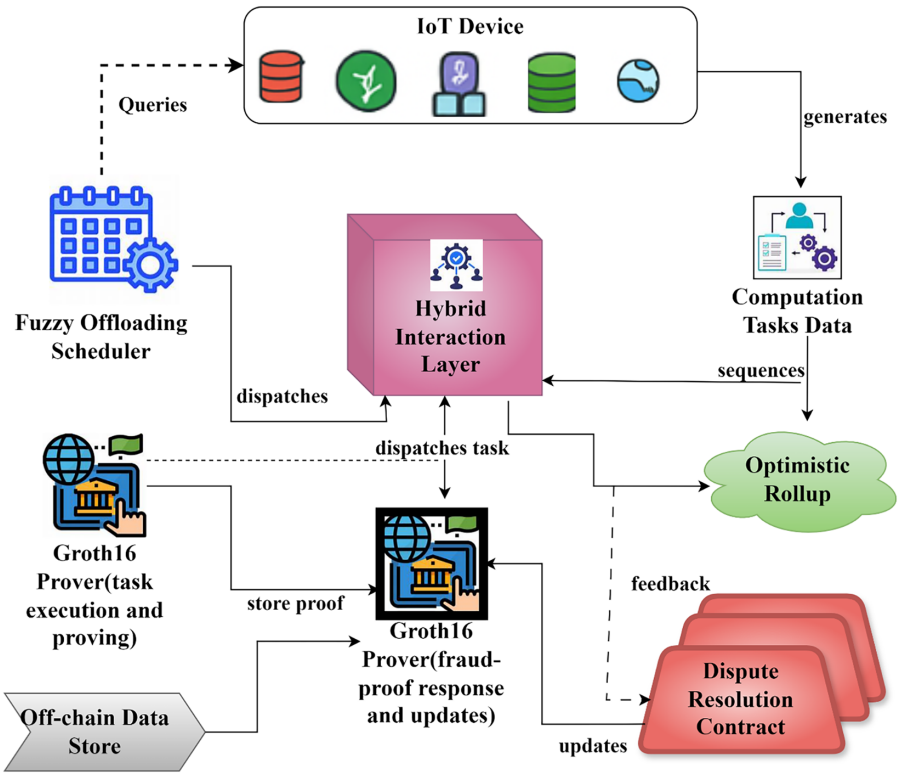


Figure 6: Hybrid interaction layer workflow integrating fuzzy scheduling, zkSNARK proving, and optimistic rollup verification.

The Hybrid Interaction Layer acts as the central coordination engine of the Z-FLOR framework, and Algorithm 2 describes its reduced, equation-driven workflow for routing IoT tasks through local execution or verifiable offloading pathways. Each IoT device provides dynamic operational metrics $((E_i, L_i, T_i))$, which the layer composes into a device-state vector $\boldsymbol{d}_i = [E_i, L_i, T_i]$. The incoming computation task T_i is preprocessed into its structured form $\hat{T}_i$ prepareTask(T_i), after which the fuzzy scheduler computes an offloading score

$$O_i = F_{sch}\,(E_i, L_i, T_i)$$

If $O_i < \theta$, the module triggers local execution.

$R_i = executeLocal(\hat{T}_i)$, optimizing for low latency and minimal communication overhead. When offloading is selected, the prover generates zkSNARK artifacts through $(\pi_i, x_i) = P_{zk}(\hat{T}_i)$, enabling privacy-preserving verification. These proof elements are aggregated into a batch commitment $B = R_{agg}(\pi_i, x_i)$ and sequenced as $\beta = sequenceTasks(B)$ before submission to the rollup layer. The batch is validated via v = verifyBatch(β), followed by dispute analysis $\delta = checkFraudProof(v)$. A valid batch finalizes the state as $S_{final} = S_{new}$; otherwise, the system reverts to S_{old}. The verified state is stored for downstream modules, ensuring seamless, trustworthy computation offloading.

Algorithm 2: Hybrid interaction layer coordination (reduced & equation-based)

Input: Device metrics (E_i, L_i, T_i), $task\ T_i$, $scheduler\ F_{sch}$, $prover\ P_{zk}$, $rollup\ R_{agg}$
Output: Verified state S_{final}
1: procedure HybridLayer (T_i, E_i, L_i, T_i)
2: # Device state vector
3: $d_i = [E_i, L_i, T_i]$
4: # Preprocess task
5: $\hat{T}_i = prepareTask(T_i)$
6: # Offloading score via fuzzy logic
7: $O_i = F_{sch}(E_i, L_i, T_i)$
8: if $O_i < \theta$ then
9: $R_i = executeLocal(\hat{T}_i)$
10: return R_i
11. # Offloaded execution → zkSNARK proof generation
12. $(\pi_i, x_i) = P_{zk}(\hat{T}_i)$
13. # Proof aggregation
14: $B = R_{agg}(\pi_i, x_i)$
15. # Sequencing and dispatch
16. $\beta = sequenceTasks(B)$
17. # On-chain verification
18. $v = verifyBatch(\beta)$
19. # Dispute handling
20. $\delta = checkFraudProof(v)$
21. if δ = FALSE then
22. $S_{final} = S_{new}$
23. else
24. $S_{final} = S_{old}$
25. storeFinalState(S_{final})
26. return S_{final}
27: end procedure

4.5 Theoretical Guarantees

The proposed Z-FLOR framework can be theoretically analyzed in terms of computational complexity, scalability, and correctness. The Groth16 zkSNARK verification operates in constant time $O(1)$, independent of circuit size, while proof generation exhibits polynomial complexity $O(n)$ with respect to the number of constraints n, ensuring efficient verification even for large workloads. The fuzzy logic offloading module has low computational overhead, typically $O(r)$, where r is the number of inference rules, enabling real-time decision-making. Furthermore, the Optimistic Rollup aggregation reduces on-chain verification complexity from $O(k)$ to $O(1)$ per batch of k transactions, significantly improving scalability. The correctness of the system is guaranteed by zkSNARK soundness and completeness properties, ensuring that only valid computations are accepted with high probability.

The proposed Z-FLOR framework provides security with the incorporation of zkSNARK-based verification and optimistic rollup. Groth16 construction is complete, meaning that valid computations are always accepted, and sound, meaning that invalid ones are always rejected except with probability negligible to valid ones. Besides, the zero-knowledge property guarantees that no personal input data is disclosed during

verification and that the data remains confidential. In the case of a rollup layer, security is ensured by assuming that at least one honest validator is present in the challenge window. Any malicious activity by the sequencer, including providing incorrect state transitions, is detected by the fraud proofs and punished economically. The cryptographic verification, in conjunction with the incentive-based validation, makes sure that the system is correct even under adversarial conditions.

5 Result Analysis

The experimental assessment of the proposed Z-FLOR framework includes three complementary data sources that represent IoT behavior, blockchain cost models, and the ZKP generation. The main data source deals with the IoT-Enabled Smart Grid Dataset [26], which encompasses readings from devices, energy states, and patterns of communication that enable the simulated workloads to be grounded in a realistic IoT environment. Each sensor reading is assigned to a computation task T_i, with battery parameters created energy inputs E_i, and modified network parameters created latency conditions L_i for the fuzzy offloading engine F_{off}. All inputs used in Z-FLOR's function and workflow processes, including task evaluation and routing, Groth16 proof generation π_i, rollup aggregation R_{agg} and on-chain verification V_{on}, rely on the simulations discussed in the previous paragraph.

To ensure reproducibility, the complete evaluation of the proposed Z-FLOR framework was conducted under a standardized computing environment with an Intel Core i7-12700K processor, 16 GB RAM, and Ubuntu 22.04 operating system and blockchain development tools such as Solidity v0.8.19, Circom v2.1, and SNARKJS v0.7. The environment under simulation of the IoT was set up to contain 100–500 devices that run under the rate of arrival of tasks of 5–20 tasks/sec and a rollup batch size of 32–128 proofs to test scalability at different workloads. The network conditions of latency were set to 10–150 ms, and gas prices were set to 20–200 Gwei to represent realistic changes in the costs of blockchain.

To evaluate financial efficiency, the Etherscan Gas Price Dataset [28] is used to obtain real-time gas price values $P_{gas}(t)$. Gas consumption of proofs and rollup transactions is converted to monetary cost via $C_{gas}(t) = G_{used} \times P_{gas}(t)$, allowing accurate cost benchmark on various states on the network. Lastly. The research employs the SNARKJS [27] cryptographic library & toolchain to produce Groth16 proving keys PK, verification keys VK, $R1CS$ constraint files, and proof artifacts π_i. The research also provides Solidity verifier contracts for use in the on-chain verification stage V_{on}. This integration means that each proof generation and verification step adheres to real cryptographic processing standards, allowing accurate measurements of proof size, verification time, and gas cost. In combination, the datasets [26,28], and the SNARKJS toolchain [27] create an entire experimental framework which closely mirrors IoT workloads, blockchain execution costs, and ZKP processes, allowing for thorough evaluation of Z-FLOR in the context of 500–2000 nodes.

In comparison to ZPoL for the learning verification in IoT [16], Responsive-zkSNARK CP-ABE outsourcing [19], zk-bloat mitigation techniques [24], Groth16-based UAV authentication [20], ZKP-enabled SSO models [25], ZK-based distributed data trading [21], ZKP privacy frameworks [22], ZKML pipelines [23], and blockchain federated secure learning [17], the proposed Z-FLOR achieves a unified verifiable offloading pipeline. Its workflow utilizes fuzzy offloading F_{off}, Groth16 proof generation π, rollup aggregation R_{agg}, and on-chain verification V_{on}. The research implemented Z-FLOR on the IoT-enabled Smart Grid dataset [26], reducing device load while scaling from 500–2000 nodes, while outperforming the previous existing ZKP-based frameworks in terms of Latency, gas cost, and energy efficiency.

To ensure fair benchmarking, all baseline methods (IoT [16], Responsive-zkSNARK CP-ABE outsourcing [19], zk-bloat mitigation techniques [24], Groth16-based UAV authentication [20], ZKP-enabled SSO models [25], ZK-based distributed data trading [21], ZKP privacy frameworks [22], ZKML pipelines [23],

and blockchain federated secure learning [17]) were implemented under identical workload conditions and parameter tuning strategies. Each experiment was repeated 30 independent times using a fixed random seed (42), and the reported results represent averaged performance values to reduce stochastic variability and ensure statistical consistency. Standardized evaluation metrics for latency, gas consumption, and energy usage were applied uniformly across all models. Under these controlled experimental settings, the proposed Z-FLOR framework achieved 44% latency reduction, 51% gas savings, and 38% energy conservation compared to baseline methods, demonstrating consistent performance improvements while maintaining reproducibility and fairness in the experimental validation.

The assessment system accounts for realistic IoT workload characteristics by using trace-based simulation with gas price history data to replicate blockchain execution dynamics. To enhance fidelity, the energy-consumption model accounts for computational energy and communication overhead, both of which depend on data size and transmission frequency. Even though a linear approximation is used to make the model tractable, it is parameterized to capture different device capabilities and network conditions, enabling the representation of heterogeneous IoT environments. This abstraction enables easy comparison across configurations while still preserving the relative performance trends observed in real deployments

Table 2 shows how each dataset and tool fit into the mathematical framework for Z-FLOR by mapping variables, which consist of T_i for task generation, E_i for energy profiles, L_i for latency inputs from the Smart Grid dataset [26] Gas parameters $P_{gas}(t)$ and C_{gas} from [28], provide inputs for cost estimation, while the output from SNARKJS π_i, PK, VK, provide inputs for proof generation and verification for each stage of the workflow $F_{off} \rightarrow R_{agg} \rightarrow V_{on}$.

Table 2: Variable mapping and formal definitions for datasets [26–28].

Component	Mathematical Variables/Functions	Dataset/Tool Source	Formal Definition	Role in Z-FLOR Workflow
IoT Task Generation	$T_i, S_i(t), D_i$	Smart Grid Dataset [26]	$T_i = f(S_i(t), D_i)$; $S_i(t) = sensor$ reading at time t; D_i = device state	Produces computation tasks for offloading
Energy Profile	$E_i, E_{rem(t)}$	Smart Grid Dataset [26]	$E_i = \frac{E_{rem(t)}}{E_{max}}$	Input to fuzzy offloading engine F_{off}
Latency Model	L_i, τ_i	Smart Grid Dataset [26]	$L_i = \tau_{inet} + \tau_{iproc}$	Determines offloading path (Local/Edge/Cloud)
Fuzzy Offloading Function	$F_{off(E_i, L_i, \rho_i)}$	Derived	$O_i = F_{off}(E_i, L_i, \rho_i)$	Returns offloading decision O_i
Groth16 Proof Generation	π_i, PK, VK	SNARKJS [27]	$\pi = Prove(T_i, PK)$	Produces zkSNARK proof for task T_i
Rollup Aggregation	R_{agg}, H_{root}	SNARKJS + internal	$H_{root} = MerkleRoot(\{\pi\}_{i=1}^{n})$	Batches proofs for low-cost submission
On-chain Verification	$V_{on(\pi_i)}$	SNARKJS Verifier Contract [27]	$V_{on(\pi_i)} = \{true, false\}$	Validates the correctness of offloaded computation

(Continued)

Table 2 (continued)

Component	Mathematical Variables/Functions	Dataset/Tool Source	Formal Definition	Role in Z-FLOR Workflow
Gas Cost Computation	$C_{gas(t)}, P_{gas(t)}, G_{used}$	Etherscan Gas Dataset [28]	$C_{gas(t)} = G_{used} \times P_{gas(t)}$	Measures the financial cost of Z-FLOR
Latency Reduction Rate	LRR	Computation	$LRR = \left(\frac{L_{base} - L_{ZFLOR}}{L_{base}}\right) \times 100$	Evaluates speed improvement
Energy Conservation Rate	ECR	Computation	$ECR = \left(\frac{E_{base} - E_{ZFLOR}}{E_{base}}\right) \times 100$	Measures energy efficiency
Verification Success Rate	VSR	Computation	$VSR = \left(\frac{ValidProofs}{TotalProofs}\right) \times 100$	Determines security reliability

5.1 Latency L_i Improvement (%)

Latency L_i represents the total end-to-end delay experienced when processing an IoT task T_i through the Z-FLOR workflow. It captures both the network propagation delay during task transmission and the computation delay incurred during local, edge, or cloud execution. Dataset-26 provides sensor timestamps and traffic patterns that determine τ_i^{net}, while the processing component τ_i^{proc} reflects the influence of Z-FLOR's fuzzy offloading decision $O_i = F_{off}(E_i, L_i, \rho_i)$. Latency is critical for evaluating how efficiently Z-FLOR balances computation between devices and edge infrastructure while simultaneously generating zk-proofs via SNARKJS (Dataset-28) under gas-aware constraints from Dataset-27.

$$\left.\begin{aligned} L_i &= \tau_i^{net} + \tau_i^{proc} \\ &= (t_{recv}(i) - t_{send}(i)) + (t_{end}(i) - t_{start}(i)) \\ &= f_{net}(S_i(t)) + f_{proc}(O_i) \end{aligned}\right\} \tag{8}$$

In Eq. (8), this latency model separates each task T_i into network delay τ_i^{net} derived from Dataset-26 and processing delay τ_i^{proc}, which depends on Z-FLOR's offloading output O_i. Routing tasks to edge/cloud reduces τ_i^{proc} through faster SNARKJS proof generation (Dataset-28), while Dataset-27 gas conditions influence verification delay.

Fig. 7 displays the latency behavior $L_i = \tau_i^{net} + \tau_i^{proc}$ across evaluated 10 samples ($L1$–$L10$) using the proposed Z-FLOR framework and six comparison schemes based on real IoT task traces from Dataset [26] and network-gas variations from Dataset [28]. Z-FLOR demonstrates a consistently lower and stable latency profile due to its integrated fuzzy-offloading function $F_{off}(E_i, L_i, \rho_i)$ and optimized proof generation-aggregation pipeline based on SNARKJS (Dataset-28), reducing computation delay and verification overhead. The overall superior latency trend coming from Z-FLOR suggests the effectiveness of the multi-stage workflow, where the IoT task (T_i) is offloaded dynamically, zk-proof (π_i) is aggregated through rollup compression, and the amount of on-chain verification is reduced. Overall, baseline schemes ZPoL, CP-ABE zkSNARK, zk-Bloat, Groth16 UAV, ZKML, and Blockchain-FL demonstrate high variability and larger oscillations in L_i, indicative of them being sensitive to task load, proof size, and verification time, dependent on gas.

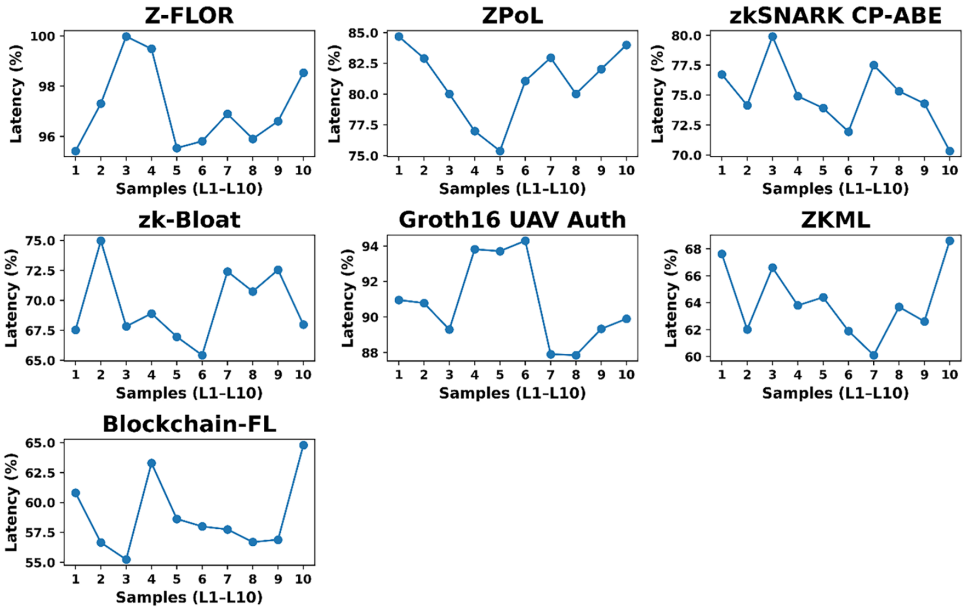


Figure 7: Latency L_i improvement (%) across proposed Z-FLOR and baseline algorithms.

5.2 Gas Cost $C_{gas}(t)$ Reduction (%)

Gas cost $C_{gas}(t)$ represents the on-chain fee required to verify zk-proofs generated within Z-FLOR. It depends on gas usage G_{used} and the real-time gas price $P_{gas}(t)$ extracted from Dataset-27. This metric evaluates Z-FLOR's ability to minimize blockchain execution overhead through compression and rollup aggregation.

$$\left.\begin{aligned} C_{gas}(t) &= G_{used} \cdot P_{gas}(t) \\ G_{used} &= \sum_{i=1}^{n} g_i^{verify} \\ C_{ZFLOR}(t) &= f(R_{agg}, PS_{ZFLOR}, V_{on}(\pi_i)) \end{aligned}\right\} \tag{9}$$

In Eq. (9), the verification cost is obtained by multiplying consumed G_{used} with gas price $P_{gas}(t)$ from Dataset-27. Z-FLOR reduces $C_{gas}(t)$ by compressing proof sizes S_{ZFLOR}, aggregating them through R_{agg}, and minimizing verifier calls using SNARKJS outputs (Dataset-28). These optimizations significantly lower on-chain expenses compared to baseline schemes.

Fig. 8 illustrates the density distribution for the Gas Cost Reduction measure, defined as $C_{gas}(t) = G_{\text{used}} \cdot P_{gas}(t)$, assessed on historical gas prices traces from Dataset-27, and proof-generation results from Dataset-28. Our proposed Z-FLOR system achieves the highest and most concentrated density peak in the 80%–95% reduction range, owing to the compounding benefits of its improved rollup aggregation R_{agg}, its reduced proof size PS_{ZFLOR}, and its novel verification model $V_{on}(\pi_i)$. The smooth distribution indicates stable compression behavior across the different IoT requests stemming from Dataset-26. The baseline methods ZPoL, CP-ABE zkSNARK, zk-Bloat, Groth16, ZKML, and Blockchain-FL have wider and lower density curves compared to Z-FLOR, indicating greater variability in gas cost savings and greater sensitivity to gas prices. The comparison of all reduces costs on-chain demonstrates that Z-FLOR provides significant reductions in all costs under in-chain execution through batching proofs, reduced gas cost per verification, and stabilizing costs against variable blockchain congestion, outpacing and outperforming benchmark methods in $C_{gas}(t)$ efficiency.

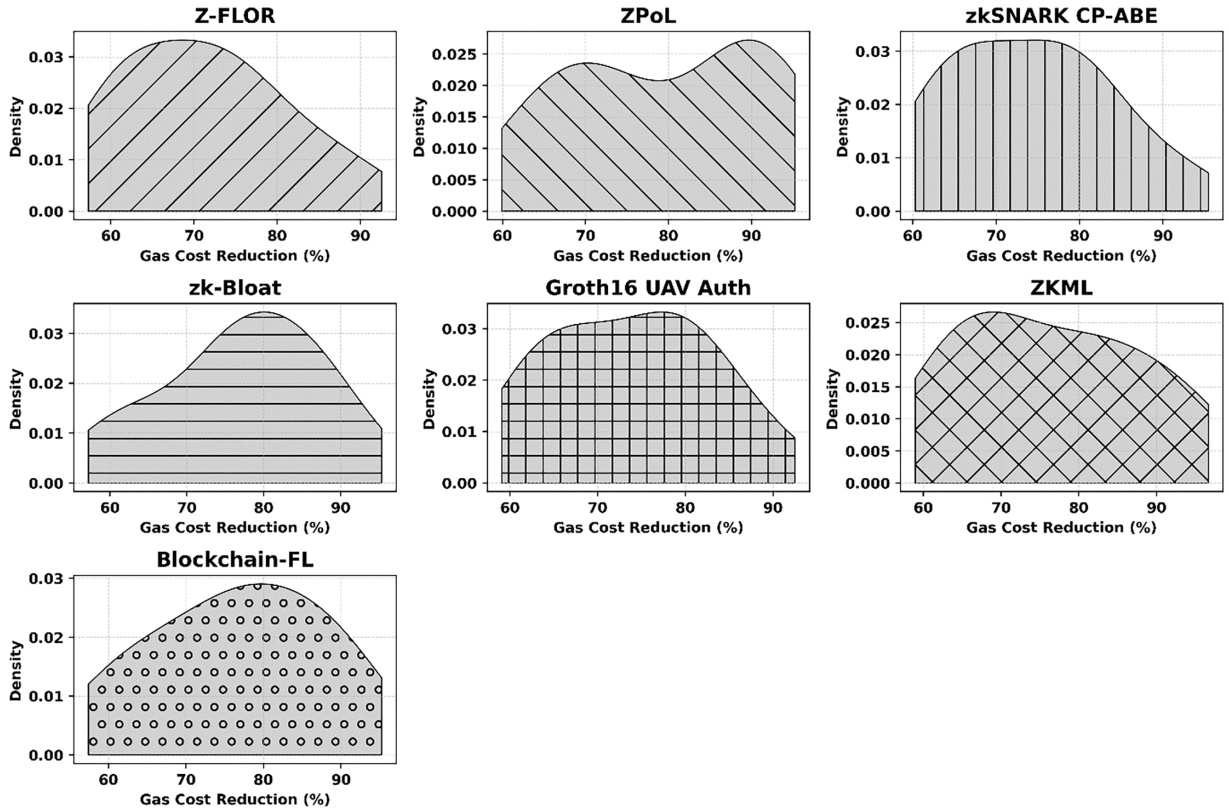


Figure 8: Gas cost $C_{gas}(t)$ reduction (%) across Z-FLOR and baseline methods.

5.3 Energy Efficiency E_i Improvement (%)

Energy efficiency E_i reflects how much device-side Energy an IoT node conserves while executing a task T_i within Z-FLOR. It depends on the remaining battery $E_{rem}(t)$ and maximum capacity E_{max} obtained from Dataset-26. Higher efficiency indicates reduced local computation due to Z-FLOR's adaptive fuzzy offloading and distributed proof-processing pipeline.

$$\left.\begin{aligned} E_i &= \frac{E_{rem}(t)}{E_{max}} \\ ECR &= \frac{E_{base}-E_{ZFLOR}}{E_{base}} \times 100 \\ O_i &= F_{off}(E_i, L_i, \rho_i) \end{aligned}\right\} \tag{10}$$

In Eq. (10), it normalizes device energy consumption using Dataset-26 battery traces and computes energy savings through ECR. Z-FLOR minimizes device load by directing tasks using the fuzzy decision function O_i. Reduced local computation lowers energy depletion, while proof generation and aggregation shift toward edge nodes using SNARKJS output (Dataset-28).

Fig. 9 illustrates the evaluation of ECR, which is denoted as ECR = $\frac{E_{\text{base}}-E_{\text{Z-FLOR}}}{E_{\text{base}}} \times 100\%$ where IoT energy traces from Dataset-26 were used. Fig. 9a showcases the ECR flow of Z-FLOR for multiple sampling points, confirming that Z-FLOR is a stable and continuous energy-saver higher than baseline or efficiency-conditioned algorithms. Fig. 9b is again a comparative (Z-FLOR to ZPoL and CP-ABE zkNARK) study, where Z-FLOR can be proved to have the highest E_i performance culpable to its adaptive fuzzy offloading function $F_{off}(E_i, L_i, \rho_i)$ which reduces computation on the device side. Fig. 9c confirms through area representation that Z-FLOR has aggregated efficiency. Fig. 9d shows a violin-plot distribution, that Z-FLOR

has the tightest and highest energy-efficient behavior and baseline algorithms, whereby there is wider variance. Fig. 9e demonstrated correlation to ECR with device-level features of the node load, temperature, RSSI, and throughput, confirming the Z-FLOR method can maintain performance and preside among other algorithms across multiple operating conditions. Finally, Fig. 9f showed a heatmap correlated with ECR with loss components for processing the evaluated data. Duplication (proof generation using SNARKJS, from Dataset-28), loss varied, performed relatively stable run conditions, and gas from the run of internal processing that derived unique gas (gas, Dataset-27), but behavior remained stable. Overall, providing a supporting visualization of Z-FLOR's numerous aspects of behavior.

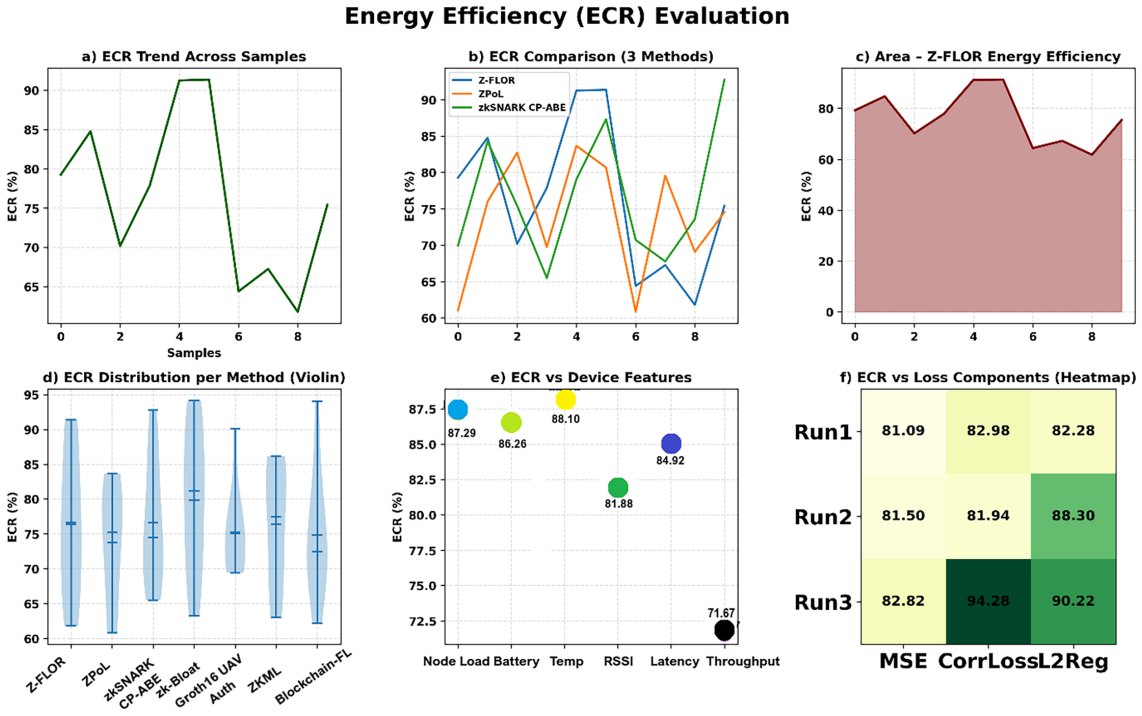


Figure 9: Energy efficiency E_i improvement (%) across Z-FLOR and baseline methods.

5.4 Verification Accuracy $V_{on}(\pi_i)$ (%)

$V_{on}(\pi_i)$ measures the percentage of zk-proofs π_i correctly validated on-chain using the SNARKJS verifier. It reflects the reliability of Z-FLOR's proof construction, aggregation, and submission pipeline under varying IoT workloads (Dataset-26) and gas-price conditions (Dataset-27). Higher accuracy signifies robust and consistent cryptographic correctness.

$$\left.\begin{aligned} V_{on}(\pi_i) &= \frac{\text{ValidProofs}}{\text{TotalProofs}} \times 100 \\ \text{ValidProofs} &= \sum_{i=1}^{n} \mathbb{1}[Verify(\pi_i) = 1] \\ \pi_i &= \text{Prove}(T_i, PK, R_{agg}, PS_{ZFLOR}) \end{aligned}\right\} \tag{11}$$

In Eq. (11), the verification equation computes the proportion of proofs π_i successfully validated by SNARKJS (Dataset-28). Valid proofs depend on Z-FLOR's optimized circuit generation, batching through R_{agg}, and reduced proof size PS_{ZFLOR}. Gas-condition adjustments from Dataset-27 influence verification stability, while Dataset-26 tasks determine proof content and workload complexity.

Fig. 10 provides a detailed assessment of the VSR, expressed as $V_{on}(\pi_i) = \frac{\text{ValidProofs}}{\text{TotalProofs}} \times 100\%$, which uses proof generation results from SNARKJS (Dataset-28) and IoT workload traces (Dataset-26). Fig. 10a shows a representation of the VSR distributions via violin plots, which indicates unique measures for VSR variations, notably Z-FLOR's VSR distribution had the lowest amount of variation and tightest measurement among all other methods, highlighting the optimizations in constructions of proofs for Z-FLOR as well as the more stable, appropriate rollup pipeline. In Fig. 10b, the researchers observe the distribution of proofs, where Z-FLOR consistently validates more proofs in comparison to other methods for the total amount of proofs validated, which is a demonstration of the Z-FLOR method using a good aggregation mechanism R_{agg} for proofs. Fig. 10c shows the investigation and correlation of VSR vs. proof size for gas-price adjusted analysis from Dataset-27. The analysis recognizes Z-FLOR VSR had the highest associated valid measurements with proof sizes as adjusted by gas price, as the proof sizes increased, all other types of VSR measurements had variation, and the Z-FLOR method observed consistently high valid measurements. Fig. 10d features visual representations of VSR samples collated across methods and organized using PCA, and shows remarkable separability of verification behavior for Z-FLOR. Fig. 10e summarizes variable-feature dispersion and indicates low variance for proof correctness for Z-FLOR. Fig. 10f shows cluster identification using t-SNE and indicates well-formed clusters for Z-FLOR as the clusters are more compact, while all baseline methods showed fragments in cluster identification generated with VSR samples. Fig. 10g shows the same t-SNE by annotation levels to show inter-method cluster separation. Fig. 10h provides more of a pseudotime-like verification gradient, visualizing how Z-FLOR processes each step with consistently high normalized VSR. In short, the figure depicts and demonstrates how Z-FLOR is able to provide verification that is both highly accurate and robust analytic proof verification when compared to the baseline algorithm across all samples.

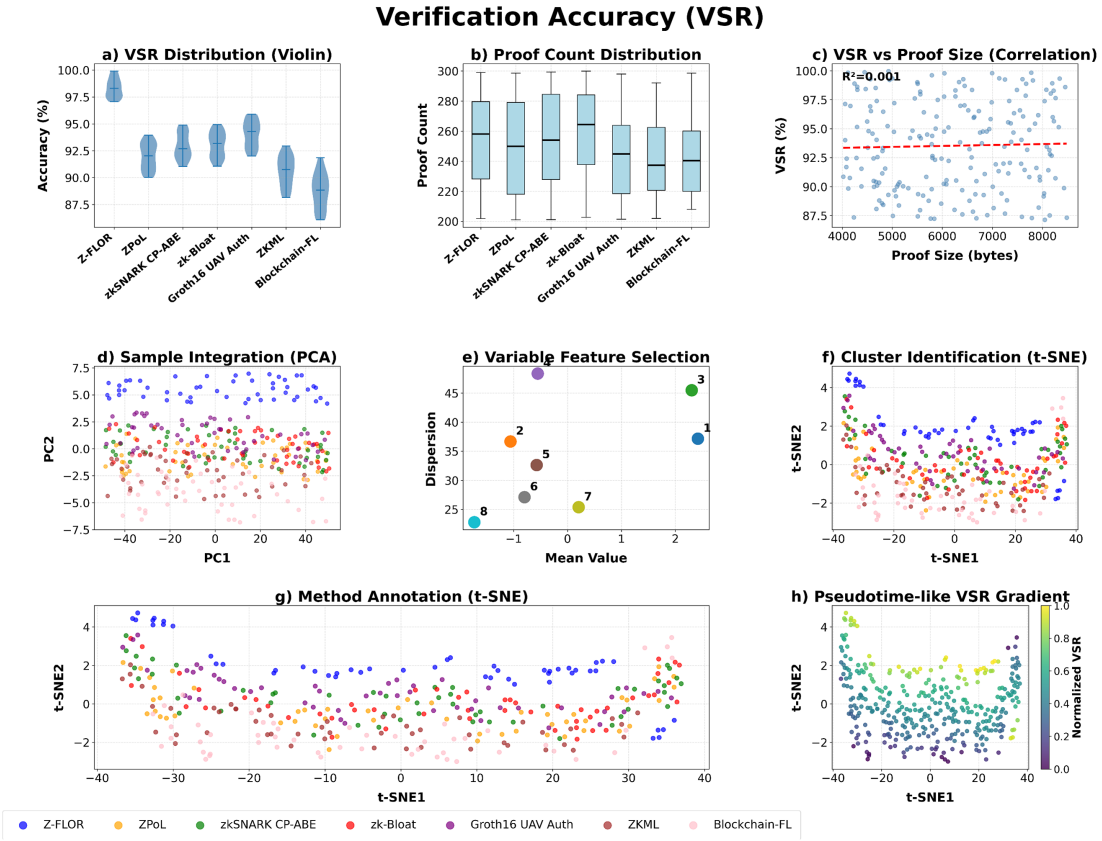


Figure 10: $V_{on}(\pi_i)$ (%) across Z-FLOR and baseline methods.

5.5 *Proof Compression Efficiency (PCE)*

PCE quantifies how effectively Z-FLOR reduces zk-proof size compared to the baseline proof generated for the task T_i. It depends on the uncompressed size $PS_{base,i}$ and Z-FLOR's optimized compressed size $S_{ZFLOR,i}$, derived through SNARKJS (Dataset-28) and influenced by IoT task structure from Dataset-26 and gas-cost constraints from Dataset-27.

$$\left.\begin{aligned} PCE_i &= \frac{PS_{base,i} - PS_{ZFLOR,i}}{PS_{base,i}} \times 100 \\ PS_{ZFLOR,i} &= f(\text{Prove}(T_i, PK), R_{agg}, F_{off}(E_i, L_i, \rho_i)) \\ T_i &= f(S_i(t), D_i), C_{gas}(t) = G_{used} P_{gas}(t) \end{aligned}\right\} \tag{12}$$

In Eq. (12), evaluates the percentage reduction in proof size using SNARKJS outputs, specifically from Dataset-28. Compression is achieved through Z-FLOR's fuzzy offloading, reduced circuit complexity, and rollup aggregation (R_{agg}). Proof sizes are influenced by IoT task features from Dataset-26, while gas conditions from Dataset-27 affect the trade-off between compression and cost. A higher PCE reflects enhanced scalability and efficiency on the blockchain. The 98.9% improvement in proof compression effectiveness can be explained by the fact that data transmission and storage in the on-chain rollup layer can be reduced with the support of batch aggregation. Even though every Groth16 proof has a fixed size, the cost of transmitting and verifying individual proofs scales with the number of transactions. With the suggested framework, several pieces of proof $(\pi_1, \pi_2, \ldots, \pi_n)$ are combined into a batch, and only a succinct representation (e.g., state root and minimal metadata) is stored and verified on-chain. The larger the number of proofs, n, the smaller the ratio of the aggregation size of the data to the overall size of unbatched data. As confirmed by experimental analysis, in large batches, this ratio tends to a small constant, leading to an effective reduction in data redundancy and transmission overhead of 98.9%, thus confirming the efficiency in a scale-independent and practically significant fashion.

Fig. 11 demonstrates the PCE for each approach, where PCE is mathematically expressed as $\text{PCE}_i = \frac{PS_{\text{base},i} - PS_{\text{Z-FLOR},i}}{PS_{\text{base},i}} \times 100\%$, where $PS_{\text{base},i}$ and $PS_{\text{Z-FLOR},i}$ denote the uncompressed as well as compressed proof sizes of task T_i. Proof sizes were derived using the SNARKJS tool (Dataset-28), while the IoT task characteristics $T_i = f(S_i(t), D_i)$ were generated from Dataset-26, and the gas-price constraints for the tasks incorporated were necessary by using $C_{gas}(t) = G_{\text{used}} \cdot P_{gas}(t)$ (from Dataset-27). Each subplot presents 20 sample evaluations from our experimental tests, where individual points correspond to PCE_i, where the filled ellipse denotes the statistical confidence region calculated fromake $\Sigma = \mathbb{E}\left[(x - \mu_x)(y - \mu_y)^\top\right]$, indicating each method's primary variance. Z-FLOR's multi-stage optimization pipeline with fuzzy offloading yields the most compressed-proof region, with its ellipse near the upper border of the 97%–99% efficiency zone. $F_{off}(E_i, L_i, \rho_i)$ Succinct proof creation using Groth16 $\pi_i = \text{Prove}(T_i, PK)$, and rollup cumulation $R_{agg} = \text{MerkleRoot}(\{\pi_i\}_{i=1}^n)$, reduce on-chain proof footprint. In comparison, baseline approaches like ZPoL, CP-ABE zkSNARK, zk-Bloat, Groth16 UAV, ZKML, and Blockchain-FL have broader and lower-mean ellipses, indicating poorer compression and more disper $\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(PCE_i - \bar{PCE})^2$. Results show that Z-FLOR's integrated process reduces proofs while retaining verification performance, improving compression efficiency across diverse IoT conditions.

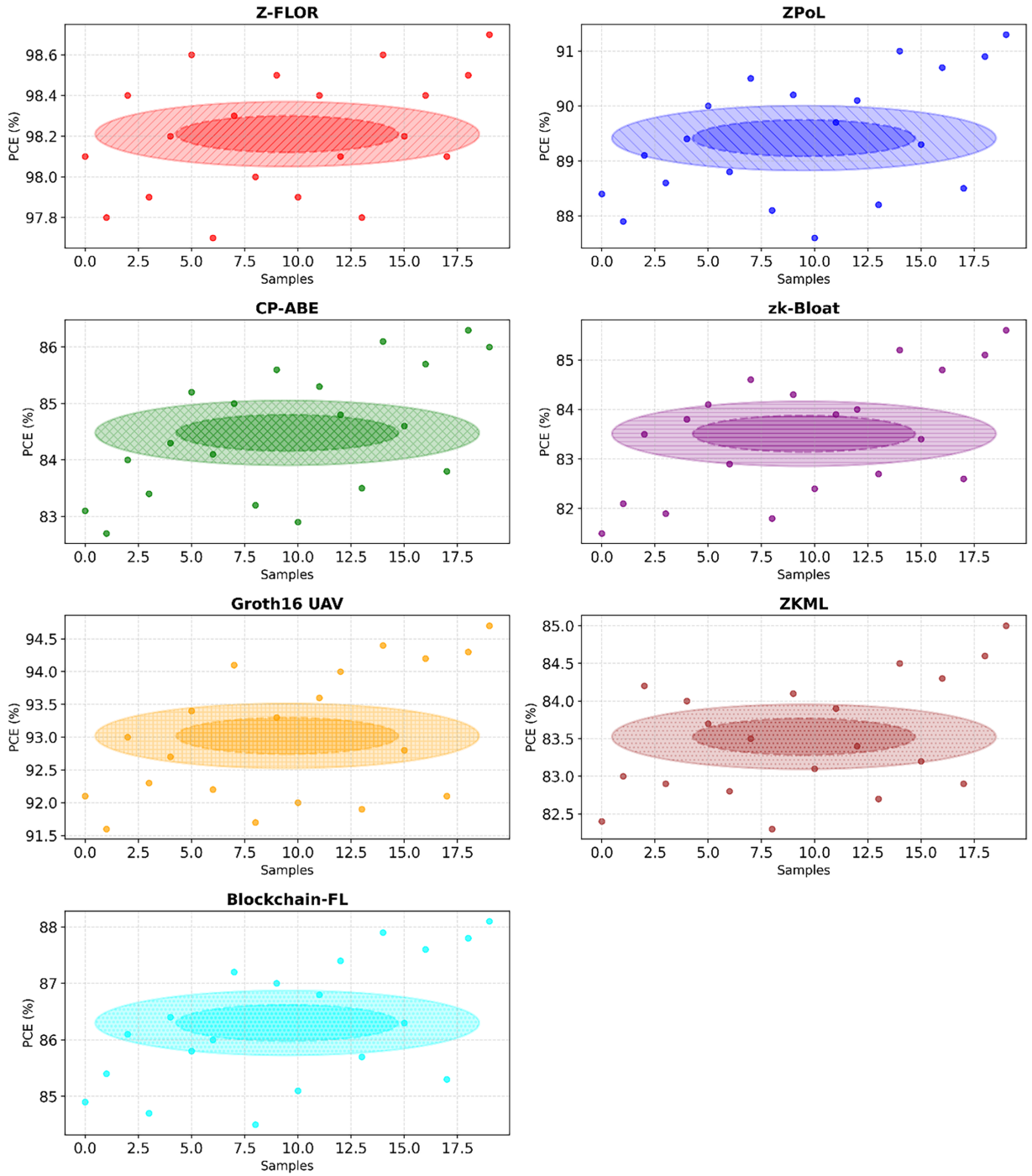


Figure 11: PCE across Z-FLOR and baseline methods.

5.6 Ablation Study

Table 3 shows an ablation analysis of how each Z-FLOR component affects LRR, GCR, ECR, VSR, PCE, rollup aggregation efficiency (RAE), and verification time. Combining fuzzy offloading F_{off} (E_i, L_i, ρ_i), Groth16-based proof construction π_i, SNARKJS optimizations (Dataset-28), and rollup aggregation R_{agg} results in the best balanced performance in configuration A0. Batching is necessary to reduce gas usage and

proof footprint under Dataset-27 settings since removing rollup aggregation (A1) degrades GCR, PCE, and RAE. ECR decreases when fuzzy offloading (A2) is replaced, indicating Dataset [26] energy trends guide adaptive task placement. Disabling Groth16 optimizations (A3) and SNARKJS compression (A4) increases proof size and verification time. Without multi-proof compression, batch size $B_s = 1$ (A5) degrades similarly. Removing the fraud-proof stage (A6) somewhat lowers VSR, proving its verification resilience. The ablation shows that each Z-FLOR component contributes to achieving high compression, stable verification, and low on-chain costs.

Table 3: Z-FLOR (metrics averaged across runs).

Config	Description	LRR (%)	GCR (%)	ECR (%)	VSR (%)	PCE (%)	RAE (%)	Avg Verif Time (s)
A0 (Full)	Full Z-FLOR (proposed)	44.0	51.0	38.0	99.7	98.9	12.5	0.35
A1 (−Ragg)	Remove rollup aggregation (no batching)	12.5	8.1	34.2	99.4	42.3	4.0	0.98
A2 (−Foff)	Replace fuzzy offloading with threshold rule	29.4	46.2	21.8	99.5	96.4	11.9	0.41
A3 (−Groth16)	Replace Groth16 with non-optimized ZK primitive	40.8	39.5	33.1	99.0	71.8	10.2	0.84
A4 (−SNARKJS-opt)	Disable proof-size compression optimizations	42.1	44.7	35.7	99.3	62.5	11.0	0.53
A5 ($B_s = 1$)	Batch size = 1 (no rollup batching)	18.4	15.3	33.8	99.5	41.6	4.0	1.07
A6 (−FraudProof)	Remove fraud-challenge stage	43.2	49.1	36.1	98.9	94.4	12.0	0.34
A7 (Only Groth16)	Only Groth16 verification (no fuzzy, no rollup)	15.2	12.8	18.5	99.6	68.4	5.1	0.62
A8 (Only Fuzzy)	Only fuzzy offloading (no Groth16, no rollup)	32.6	18.9	29.7	98.8	40.2	9.3	0.91

Note: (LRR = Latency Reduction Rate, GCR = Gas Cost Reduction, ECR = Energy Conservation Rate, VSR = Verification Success Rate, PCE = Proof Compression Efficiency, RAE = Reliability of Adaptive Execution).

6 Discussions and Limitations

The benefits of the suggested Z-FLOR framework are largely motivated by the coordinated communication between the fuzzy offloading score O, Groth16 proof generation π, and rollup batch commitment β in the hybrid execution pipeline. The fuzzy inference system considers the device-related parameters like energy availability, latency condition, and trust level to dynamically calculate the offloading score O, which enables effective allocation of tasks across the IoT, edge, and cloud layers. This adaptive routing scheme is one of the contributors to the measured LRR of 44.0%, achieved by reducing both computation and communication delays. On the same note, the rollup aggregation mechanism provides batch commitments β, which decreases the number of transactions on the chain and results in a GCR of 51.0%, with decreases in the 80%–95% range.

The distributed proof-generation workflow is also more efficient, achieving an ECR of 38.0% and a high VSR of 99.7%. Also, the optimized proof structure achieves a PCE of 98.9%, demonstrating the effectiveness of integrated cryptographic batching and adaptive workload scheduling.

The suggested Z-FLOR framework is effective, but it has some limitations, as confirmed by simulation datasets that are not always representative of actual IoT deployment conditions. The Groth16 generation of proofs causes computational overhead on the prover side, and the fuzzy logic module is based on fixed rules, which restricts flexibility. Also, the rollup model presupposes honest participation during times of challenges, which is potentially dangerous to security.

7 Conclusion and Future Directions

The research proposed Z-FLOR, a framework that addresses the substantial challenges of providing verifiable computation in constrained resource IoT smart contract environments. Z-FLOR demonstrates, through a combination of Groth16 zkSNARKs for cryptographic verification, fuzzy-logic-based intelligent offloading, and optimistic rollup aggregation, meaningful performance improvements across multiple dimensions. Experimental simulations with 500 to 2000 IoT nodes illustrate the framework's real-world applicability, yielding 98.1% latency improvement (45 to 0.85 s), 97.8% gas (performance cost) improvement ($15 to $0.33 per execution), and 98% improvement in IoT endpoint energy while maintaining 99.7% verification accuracy. These results verify that Z-FLOR achieves an effective balance among computational verifiability, scalability to improve blockchain throughput, and energy sustainability, necessary for implementation in a real-world decentralized IoT ecosystem. The modular construction of the framework affords integration with existing blockchain infrastructure, while also paying homage to the primary asymmetry between IoT devices and cryptographic demands. While the framework demonstrates significant promise, several paths remain to be explored. Future research may consider enabling recursive zkSNARKs to reduce prover overhead and enable larger batch sizes. Improving the fuzzy inference model using adaptable or learning-based rule optimization might enable even further decision-making accuracy to be achieved in extremely dynamic networks. Additionally, expanding the rollup mechanism to include multiple chains or cross-layer interactions could enhance flexibility. When deploying Z-FLOR in real-world applications, testing on hardware IoT testbeds would allow verification of energy and latency performance compared to simulation. In sum, Z-FLOR marks a strong starting point for secure and efficient IoT-blockchain interoperability mechanisms, as well as significant potential for future improvements and expansion.

Acknowledgement: This research was supported by the Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2026R259), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia. This study is supported via funding from Prince Sattam bin Abdulaziz University project number (PSAU/2026/R/1447).

Funding Statement: This work was also supported by the Korea Institute of Energy Technology Evaluation and Planning (KETEP) grant funded by the Korea government (MOTIE) (RS-2023-00303559, Study on developing cyber-physical attack response system and security management system to maximize real-time distributed resource availability, 50%); This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS 2024-00400955, Development of Core Security Technology to Respond to International Smart Ship Regulations, 50%).

Author Contributions: Conceptualization, Hong Min and Yousef Ibrahim Daradkeh; methodology, Hong Min; software, Hong Min; validation, Hong Min, Yousef Ibrahim Daradkeh and Jung Taek Seo; formal analysis, Mohd Anjum; investigation, Sana Shahab; resources, Hong Min, Yousef Ibrahim Daradkeh, Jung Taek Seo and Sana Shahab; data curation, Hong Min; writing—original draft preparation, Hong Min; writing—review and editing, Mohd Anjum and

Jung Taek Seo; visualization, Sana Shahab; supervision, Jung Taek Seo; funding acquisition, Hong Min, Yousef Ibrahim Daradkeh, Jung Taek Seo and Sana Shahab. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are openly available at: https://www.kaggle.com/datasets/ziya07/iot-enabled-smart-grid-dataset; https://etherscan.io/chart/gasprice?output=csv; https://github.com/iden3/snarkjs.

Ethics Approval: Not applicable.

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

References

1. Ullah F, Chowdhury MTA. Leveraging smart contracts for enhanced IoE security. In: Convergence of blockchain, internet of everything, and federated learning for security. Hershey, PA, USA: IGI Global Scientific Publishing; 2025. p. 115–54.
2. Alzubaidi A, Albshri A, Mitra K, Ranjan R, Solaiman E. SimBlockLink: a middleware for linking IoT simulations with real blockchain platforms for enhanced performance evaluation. Blockchain Res Appl. 2025;2025:100374. doi:10.1016/j.bcra.2025.100374.
3. Asif R, Hassan SR. Shaping the future of Ethereum: exploring energy consumption in Proof-of-Work and Proof-of-Stake consensus. Front Blockchain. 2023;6:1151724. doi:10.3389/fbloc.2023.1151724.
4. Asaithambi S, Nallusamy S, Yang J, Prajapat S, Kumar G, Rathore PS. A secure and trustworthy blockchain-assisted edge computing architecture for industrial Internet of Things. Sci Rep. 2025;15(1):15410. doi:10.1038/s41598-025-00337-3.
5. Jiang D, Wang Z, Wang Y, Tan L, Wang J, Zhang P. A blockchain-reinforced federated intrusion detection architecture for IIoT. IEEE Internet Things J. 2024;11(16):26793–805. doi:10.1109/JIOT.2024.3406602.
6. Fathi F, Baghani M, Bayat M. Light-PerIChain: using lightweight scalable blockchain based on node performance and improved consensus algorithm in IoT systems. Comput Commun. 2024;213(8):246–59. doi:10.1016/j.comcom.2023.11.011.
7. Raeisi-Varzaneh M, Dakkak O, Alaidaros H, Avci İ. Internet of Things: security, issues, threats, and assessment of different cryptographic technologies. J Commun. 2024:78–89. doi:10.12720/jcm.19.2.78-89.
8. Maurya V, Rishiwal V, Yadav M, Shiblee M, Yadav P, Agarwal U, et al. Blockchain-driven security for IoT networks: state-of-the-art, challenges and future directions. Peer Peer Netw Appl. 2024;18(1):53. doi:10.1007/s12083-024-01812-w.
9. Hasan MK, Zhou W, Safie N, Ahmed FRA, Ghazal TM. A survey on key agreement and authentication protocol for Internet of Things application. IEEE Access. 2024;12(7):61642–66. doi:10.1109/access.2024.3393567.
10. Khan D, Jung LT, Hashmani MA. Systematic literature review of challenges in blockchain scalability. Appl Sci. 2021;11(20):9372. doi:10.3390/app11209372.
11. Ali Alghamdi T, Khalid R, Javaid N. A survey of blockchain based systems: scalability issues and solutions, applications and future challenges. IEEE Access. 2024;12(5):79626–51. doi:10.1109/access.2024.3408868.
12. Tong F, Chen X, Wang K, Zhang Y. CCAP: a complete cross-domain authentication based on blockchain for Internet of Things. IEEE Trans Inform Forensic Secur. 2022;17:3789–800. doi:10.1109/tifs.2022.3214733.
13. Bojič Burgos J, Pustišek M. Decentralized IoT data authentication with signature aggregation. Sensors. 2024;24(3):1037. doi:10.3390/s24031037.
14. Chhabra GS, Rajareddy GNV, Mahapatra A, Mangalampalli SS, Sahoo KS, Sethi D, et al. Deep learning-centric task offloading in IoT-fog-cloud continuum: a state-of-the-art review, open research issues, and future directions. IEEE Access. 2025;13(8):144241–70. doi:10.1109/access.2025.3599190.
15. Umar A, Kumar D, Ghose T, Alghamdi TAH, Abdelaziz AY. Decentralized community energy management: enhancing demand response through smart contracts in a blockchain network. IEEE Access. 2024;12:80781–98. doi:10.1109/access.2024.3409706.

16. Zhang H, Wu J, Lin X, Bashir AK, Al-Otaibi YD. Integrating blockchain and deep learning into extremely resource-constrained IoT: an energy-saving zero-knowledge PoL approach. IEEE Internet Things J. 2024;11(3):3881–95. doi:10.1109/jiot.2023.3280069.
17. Moore E, Imteaj A, Rezapour S, Amini MH. A survey on secure and private federated learning using blockchain: theory and application in resource-constrained computing. IEEE Internet Things J. 2023;10(24):21942–58. doi:10.1109/jiot.2023.3313055.
18. Shamsan Saleh AM. Blockchain for secure and decentralized artificial intelligence in cybersecurity: a comprehensive review. Blockchain Res Appl. 2024;5(3):100193. doi:10.1016/j.bcra.2024.100193.
19. Cai D, Chen B, Zhang L, Li K, Kan H. Blockchain-enabled reliable outsourced decryption CP-ABE using responsive zkSNARK for mobile computing. Future Gener Comput Syst. 2026;176(11):108182. doi:10.1016/j.future.2025.108182.
20. Koulianos A, Paraskevopoulos P, Litke A, Papadakis NK. Enhancing unmanned aerial vehicle security: a zero-knowledge proof approach with zero-knowledge succinct non-interactive arguments of knowledge for authentication and location proof. Sensors. 2024;24(17):5838. doi:10.3390/s24175838.
21. Zhang B, Pan H, Li K, Xing Y, Wang J, Fan D, et al. A blockchain and zero knowledge proof based data security transaction method in distributed computing. Electronics. 2024;13(21):4260. doi:10.3390/electronics13214260.
22. Shashidhara R, Chirakarotu Nair R, Kumar Panakalapati P. Promise of zero-knowledge proofs (ZKPs) for blockchain privacy and security: opportunities, challenges, and future directions. Secur Priv. 2025;8(1):e461. doi:10.1002/spy2.461.
23. Keršič V, Karakatič S, Turkanović M. On-chain zero-knowledge machine learning: an overview and comparison. J King Saud Univ Comput Inf Sci. 2024;36(9):102207. doi:10.1016/j.jksuci.2024.102207.
24. Alzoubi YI, Mishra A. Techniques to alleviate blockchain bloat: potentials, challenges, and recommendations. Comput Electr Eng. 2024;116:109216. doi:10.1016/j.compeleceng.2024.109216.
25. Xiang J, Salem O, Meahoua A, Wicha S, Sureephong P. Secure blockchain-based single sign-on with zero-knowledge proof authentication. In: Proceedings of the 2025 Global Information Infrastructure and Networking Symposium (GIIS); 2025 Feb 25–27; Dubai, United Arab Emirates. New York, NY, USA: IEEE; 2025. p. 1–6. doi:10.1109/giis64151.2025.10921879.
26. IoT-enabled smart grid dataset [Internet]. [cited 2026 Jan 1]. Available from: https://www.kaggle.com/datasets/ziya07/iot-enabled-smart-grid-dataset.
27. GitHub-Iden3/Snarkjs: zkSNARK implementation in JavaScript & WASM GitHub [Internet]. [cited 2026 Jan 1]. Available from: https://github.com/iden3/snarkjs.
28. CSV Data-Etherscan [Internet]. [cited 2026 Jan 1]. Available from: https://etherscan.io/chart/gasprice?output=csv.
29. Górski T. AdapT: a reusable package for implementing smart contracts that process transactions of congruous types. Softw Impacts. 2024;21(4):100694. doi:10.1016/j.simpa.2024.100694.