IoT

MDPI

Article

A Blockchain and Federated Learning Framework for Image-Based IoT Malware Detection and Prevention

Najem N. Sirhan [1,*], Riyad Alrousan [2] and Hussam N. Fakhouri [3]

[1] Computer Science Department, Faculty of Information Technology, University of Petra, Amman 11196, Jordan
[2] Department of Design and Visual Communication, School of Architecture and Built Environment (SABE), German Jordanian University (GJU), Amman 11180, Jordan; riyad.alrousan@gju.edu.jo
[3] Faculty of Artificial Intelligence, Al-Balqa Applied University, As-Salt 19117, Jordan; h.fakhouri@bau.edu.jo
* Correspondence: najem.sirhan@uop.edu.jo

Abstract

Internet of Things (IoT) devices are increasingly targeted by rapidly evolving malware, yet collaborative detection remains challenged by privacy leakage, noisy and imbalanced training data, and weak integrity guarantees when sharing model updates. This paper presents *Mal-Fedchain*, a secure and privacy-preserving framework for image-based IoT malware detection and prevention that couples federated learning with blockchain and honeypot-assisted behavioral monitoring, targeting Linux-capable IoT gateway devices. Portable Executable (PE) binaries are transformed into grayscale images using a corrected fixed-width byte-mapping pipeline stabilized by an information-maximizing GAN (IM-GAN). A bi-level preprocessing pipeline applies two-sided weighted sparse representation (T-WSR) denoising—designed to selectively suppress zero-padding artifacts, high-entropy packed regions, and sparse opcode noise while preserving discriminative section-boundary texture—followed by geometric augmentation to mitigate class imbalance. Malware detection and family attribution are performed using a residual capsule-based network (RBCN) that fuses discriminative visual representations with PE-header features via concatenation, improving robustness against polymorphism and obfuscation. A formal threat model governs three adversary classes: a semi-honest aggregation server, a bounded fraction of malicious clients (up to 30%), and a passive eavesdropper. To enable collaboration without exposing raw data, clients train locally and share only MemCbar-encrypted updates; a permissioned Hyperledger Fabric blockchain ledger records hashed updates and security events to provide integrity, traceability, and tamper resistance. A file-system-integrated honeypot captures evasive behaviors and logs auditable evidence to strengthen prevention. Experiments on the Malimg dataset across five ablation configurations demonstrate that the corrected RBCN pipeline achieves 93.52% accuracy, 92.40% precision, 93.52% recall, 92.52% F-measure, MCC of 0.9245, and AUC of 0.9976 in its centralized configuration, and 65.62% accuracy with AUC of 0.9840 in the full federated configuration with five clients and eight communication rounds, substantially outperforming all baselines across all reported metrics.

Keywords: federated learning; malware visualization; IoT security; blockchain; capsule network; privacy-preserving machine learning; honeypot; grayscale image classification

Academic Editor: Amiya Nayak

Received: 23 April 2026
Revised: 3 July 2026
Accepted: 7 July 2026
Published: 9 July 2026

1. Introduction

Cyberattackers continue to proliferate malware variants across Internet-connected devices [1]. Malware (i.e., *malicious software*) is engineered to gain unauthorized access

to systems in order to steal information, disrupt operations, or compromise computing environments [2]. As the volume of malicious files grows rapidly, adversaries increasingly embed harmful code into executables, resulting in a wide spectrum of threats—including spyware, ransomware, adware, keyloggers, rootkits, botnets, Trojans, and worms [3,4].

This threat is particularly acute in the Internet of Things (IoT), where heterogeneous devices are interconnected through wireless networks and widely deployed in smart homes and hospitals, energy management, smart grids, and industrial systems [5]. Many IoT devices operate under strict constraints on security, storage, and computation, making them attractive targets for large-scale exploitation and botnet recruitment (e.g., Mirai) [6]. Once compromised, such devices can be leveraged to disrupt services and propagate attacks across the broader network.

To counter these risks, malware detection and classification methods commonly rely on static or dynamic analysis. Static analysis inspects binaries without execution and is often suitable for large-scale screening using features such as byte sequences, strings, opcodes, and permissions [7]. Dynamic analysis, in contrast, executes samples in controlled environments to observe behavioral traces [8]. Despite its advantages, dynamic analysis can be computationally expensive, time-consuming, and sometimes ineffective against sophisticated malware that conceals or delays key behaviors; obfuscation further complicates reliable detection [9]. These challenges have motivated learning-based approaches that can improve robustness and automate feature extraction.

Recently, machine learning and deep learning have become central to modern detection pipelines [10]. Deep learning models, in particular, can learn discriminative representations directly from raw or lightly processed inputs and have shown strong performance in malware classification [11]. Malware visualization complements these approaches by converting binaries into images (e.g., grayscale, Markov, or RGB), enabling both human inspection and image-based learning models to capture structural signatures without executing the code [12]. This binary-to-image approach offers three principal advantages: (i) it avoids the computational cost and safety risks of dynamic execution; (ii) structural signatures introduced by shared compilers, packers, or malware-generation toolkits are often visually distinguishable as texture, enabling family-level discrimination even under minor code modifications; and (iii) the resulting image representation is directly compatible with mature convolutional and capsule-based architectures developed for image classification. At the same time, the approach has important limitations: encrypted or heavily packed binaries exhibit near-uniform, high-entropy byte distributions that produce visually similar images regardless of the underlying malicious logic, and the conversion discards semantic control-flow information. These limitations motivate the complementary use of structured PE-header features and honeypot-derived behavioral evidence in *Mal-Fedchain* so that the framework does not rely on visual texture alone when it is ambiguous.

It is also important to clarify the target deployment context. *Mal-Fedchain* is designed for *Linux-capable* IoT devices that can host a Python runtime and support local model training, such as single-board computers (e.g., Raspberry Pi 3/4, NVIDIA Jetson Nano) and industrial IoT gateways. Bare-metal microcontrollers and pure real-time OS nodes are out of scope for the local training role, though they may participate as monitored endpoints forwarding data to a capable gateway client for analysis.

However, three interconnected limitations remain prominent in IoT-oriented malware detection: (i) insufficient security and privacy guarantees in collaborative settings (e.g., plaintext sharing of outputs or model updates, which can enable inference or poisoning attacks) [11,13,14]; (ii) degraded classification performance when datasets contain noise, irrelevant artifacts, or class imbalance [15,16]; and (iii) limited preventive capabilities against evasive malware that employs polymorphism or obfuscation to bypass detection [4,6,17,18].

Motivated by these gaps, we propose *Mal-Fedchain*, a framework that integrates malware visualization and enhanced deep learning with federated learning, blockchain-based integrity, and honeypot-assisted prevention to support privacy-preserving and secure malware detection in IoT environments. In particular, this work:

- Converts Portable Executable (PE) files into grayscale images using a corrected fixed-width byte-mapping pipeline (Algorithm 1, $W = 256$) to enable image-based malware classification without executing potentially harmful code;
- Improves learning quality through semantically grounded noise reduction (T-WSR) and data augmentation to reduce the impact of artifacts and class imbalance on classification performance;
- Performs malware detection and family attribution using a residual capsule-based network (RBCN) that preserves part–whole spatial relationships through dynamic routing and fuses grayscale image features with structured PE-header fields, improving robustness against polymorphic and obfuscated variants;
- Incorporates a honeypot mechanism to attract and observe adversarial activity, strengthening preventive capability and situational awareness; and operates under a formal threat model comprising a semi-honest aggregation server, a bounded fraction of malicious clients, and a passive eavesdropper;
- Enables collaborative learning via federated learning while using MemCbar-encrypted updates and a blockchain layer (Hyperledger Fabric, Raft consensus) to support trusted coordination and integrity protection for exchanged updates.

The proposed *Mal-Fedchain* framework is evaluated across five ablation configurations on the Malimg dataset using accuracy, precision, recall, F-measure, Matthews Correlation Coefficient (MCC), and Area Under the ROC Curve (AUC), alongside empirical security metrics (gradient inversion resistance and Byzantine poisoning robustness). The corrected centralized RBCN achieves 93.52% accuracy and AUC of 0.9976; the full federated configuration achieves 65.62% accuracy and AUC of 0.9840 with five clients and eight communication rounds.

The remainder of this paper is organized as follows. Section 2 reviews related work on malware detection and classification and summarizes key limitations. Section 3 defines the threat model, specifies the target IoT node types, and presents the Mal-Fedchain system model and overall architecture, including binary-to-image conversion, preprocessing, and the federated learning workflow. Section 4 presents the experimental setup—including the federated learning configuration and ablation study design—and reports comparative results with accuracy, precision, recall, F-measure, MCC, AUC, and security metrics. Finally, Section 5 concludes the paper and outlines future research directions.

2. Literature Survey

Malware visualization has been widely adopted to enable learning models to capture structural patterns in binaries without executing potentially harmful code. For example, Ref. [11] converts malware binaries into color images and classifies them using an enhanced convolutional neural network, supported by normalization to highlight obfuscated/encrypted samples and augmentation to reduce class imbalance (evaluated on Malimg and ImageNet). While the approach demonstrates the practicality of image-based learning, it does not explicitly incorporate security protections for exchanging results or artifacts, leaving the pipeline susceptible to tampering and reducing robustness in adversarial settings.

To strengthen trust and integrity during file sharing and verification, blockchain has been explored as a security layer. In [19], blockchain is used to distribute and validate executable hashes: nodes scan an advertised file, record malicious hashes as blockchain

transactions, and consult the ledger to decide whether a file should be treated as benign or malicious. This design improves traceability and verification; however, it largely treats blockchain as a logging/verification mechanism and does not integrate AI-based models that can adapt to evolving malware behavior, which can constrain detection capability against rapidly changing threats.

Privacy-preserving collaboration is another major direction, particularly for IoT environments where data are distributed across many owners and devices. In [13], federated learning is used to study IoT malware and poisoning-related threats by training supervised and unsupervised models (e.g., multilayer perceptrons and autoencoders) on locally collected network-traffic data, with aggregation performed at a central server and evaluation on N-BaIoT. Although this setup reduces raw-data exposure, client-side information is still shared with the server without encryption, creating opportunities for privacy leakage and manipulation during communication.

Beyond the learning paradigm, several studies focus on how malware is represented and how features are extracted from those representations. In [20], IoT malware diversity is analyzed using a CNN pipeline that includes preprocessing into RGB images, attention, CNN-based feature extraction, and spatial pooling to normalize dimensionality. While attention can improve discriminative learning, RGB representations may increase preprocessing complexity and introduce ambiguity in robust color construction, which can affect analysis consistency across diverse binaries. In contrast, Ref. [21] simplifies representation by converting executables into grayscale images (via 8-bit encoding and reshaping) and applies CNN-based classification with normalized image sizes and feature extraction over multiple texture scales using the Microsoft malware dataset. Nevertheless, CNN-based systems can overfit and may be sensitive to small geometric perturbations unless robustness mechanisms (e.g., augmentation and regularization strategies) are carefully incorporated.

Other works propose alternative image constructs and lightweight designs to improve efficiency while retaining discriminative power. In [17], binaries are mapped to Markov images—two-dimensional representations in which each cell (i, j) encodes the transition probability (or frequency) of byte value i being immediately followed by byte value j in the binary stream, producing a 256×256 matrix that captures local statistical dependencies in the byte sequence and can be visualized as a grayscale image [17]—and classified with a lightweight CNN enhanced by channel muting and deeper convolution, including multidimensional representations formed by combining multiple Markov images. Despite promising results, the focus remains primarily on learning known characteristics and does not sufficiently emphasize preventive measures for modified or polymorphic malware. Similarly, Ref. [18] avoids explicit image conversion by using bit- and byte-level representations: byte sequences are compressed to fixed lengths and processed with a one-dimensional CNN on Malimg and Microsoft datasets. While effective for known patterns, such schemes can be bypassed when adversaries transform code structures to evade learned signatures.

Transfer learning, attention, and hybrid feature engineering have also been used to improve generalization across families. In [22], malware images derived through wavelet transforms are classified using a CNN with spatial attention and transfer learning (Malimg), with grayscale-to-RGB conversion used to leverage pretrained models; however, augmentation is not emphasized, which can limit robustness under data imbalance. In [23], PE visualization is combined with extracted Haralick and string features, followed by traditional classifiers (e.g., XGBoost, SVM, and random forest), where random forest reportedly achieves the best performance; yet, visualized binaries without adequate denoising and artifact removal can still degrade accuracy. Ensemble strategies have also been explored: Ref. [24] transfers pretrained CNN backbones (e.g., ResNet50 and VGG16) to the malware domain and applies PCA for dimensionality reduction before classification, but the de-

sign does not emphasize logging and reuse of discovered malware patterns, potentially increasing repeated processing overhead. For IoT-specific constraints, Ref. [25] proposes a lightweight detection framework using grayscale visualization with neural networks (e.g., DenseNet and CNN) and attention mechanisms trained on Malimg and BIG2015; however, assuming benign files remain clean can be risky when polymorphic malware mimics benign behavior. Finally, Ref. [26] combines multiple CNNs (DenseNet, ResNet, and MobileNet) with random-forest voting on RGB representations (MaleVis), but centralizing intrusion-detection information can expose the system to manipulation and further increase computational complexity.

Overall, existing IoT malware-detection studies achieve strong performance by combining binary-to-image (or byte-level) representations with deep learning and, in some cases, blockchain or federated learning [11,13,17–26]. However, recurring gaps remain: (i) security and privacy weaknesses when outputs or model updates are exchanged in plaintext (especially in collaborative learning) [11,13–16]; (ii) insufficient preprocessing and limited augmentation, which amplifies noise, imbalance, and false positives [15,16,22,23,27]; (iii) computational overhead and model fragility in resource-constrained IoT deployments [15,21,24]; (iv) limited preventive capability against evasion and polymorphism [17,18,25]; and (v) architectural risks introduced by centralization or by using blockchain without tight AI integration [19,26]. Motivated by these limitations, Mal-Fedchain combines (a) efficient grayscale visualization (IMGAN) and robust learning (RBCN), (b) a bi-level preprocessing pipeline with denoising (T-WSR) followed by targeted augmentation, (c) privacy-preserving federated learning with encrypted update exchange (MemCbar), (d) blockchain-based integrity and traceability to reduce manipulation risk and enable trusted coordination, and (e) honeypot-assisted prevention to improve resilience against modified and evasive malware.

Prior studies have advanced image-based and collaborative malware detection for IoT systems; however, several persistent gaps continue to limit robustness, privacy, and practical deployment. In [15], malware binaries are visualized and processed using Gabor-filter feature extraction followed by a three-layer CNN trained on BIG2015 and Malimg, yet the approach introduces high computational overhead and may produce redundant features that inflate latency and can even degrade accuracy. Moreover, detection relies on sharing users' device data in raw form, exposing privacy in the absence of security safeguards, while the lack of explicit noise handling can propagate artifacts into training and increase false positives. Similar weaknesses appear in [16], where binaries are transformed into 8-bit vectors to form one-dimensional representations for neural-network classification on Microsoft and Malimg datasets: noisy representations may be fed directly to the classifier, and the pipeline does not provide strong security guarantees against manipulation. Although transfer-learning-based methods in [27] leverage models such as Inception V3 and AlexNet with augmentation for grayscale feature extraction, augmentation without denoising can amplify noise and elevate false positives, and conventional classifiers may show limited robustness and interpretability when data are not sufficiently representative. Finally, Ref. [14] proposes a federated learning scheme using LSTM/RNN models with blockchain storage of outcomes, but transmitting local outputs without encryption creates opportunities for tampering; APK decompilation and obfuscation can delay response while attacks persist; and LSTM/RNN training complexity and overfitting can reduce accuracy in resource-constrained settings. Collectively, these works highlight a common set of problems: computational inefficiency, privacy leakage through insecure communication, noise-aware learning deficiencies, and limited preventive capability against polymorphic or evasive malware.

More recent work (2022–2024) has addressed some of these gaps but continues to leave room for the integrated approach proposed here. Ref. [28] introduced FedAvg, the foundational aggregation algorithm for federated learning, which has since been widely adopted as the standard baseline in distributed malware detection pipelines. Building on FedAvg, Ref. [29] proposed FedProx, which adds a proximal regularization term to the local objective to improve convergence under system and data heterogeneity—a property particularly relevant to IoT environments where clients differ significantly in hardware capability and local dataset size. Despite these advances, neither FedAvg nor FedProx incorporates update encryption or blockchain-based audit mechanisms, leaving the aggregation process exposed to a semi-honest server and Byzantine client attacks. A representative recent work from IEEE Transactions on Information Forensics and Security [30] proposes FEDriod, a comprehensive federated learning framework for Android malware detection that employs genetic evolution strategies to simulate malware variant generation and achieves strong cross-dataset generalization. While FEDriod demonstrates the potential of FL for malware family detection, it targets Android APK binaries on the Drebin and CIC datasets and does not address PE-binary visualization, blockchain-based integrity, or honeypot-assisted prevention. Direct numerical comparison with FEDriod is therefore not possible due to the different malware platforms and datasets, but it serves as a reference point for the current state of FL-based malware detection in the 2022–2024 literature. Addressing the IoT-specific threat landscape, recent work on federated malware detection for IoT [11] demonstrates that combining image-based binary representations with distributed learning can achieve competitive detection rates while preserving data locality. However, such methods still lack a unified framework that simultaneously addresses update privacy (encryption), integrity verification (blockchain), and proactive threat capture (honeypot). Table 9 in Section 4.5 summarises these limitations alongside the direct baselines used in our comparative evaluation.

To address these limitations, we propose an image-based malware classification and prevention framework that integrates robust learning with privacy-preserving and security-enhancing mechanisms. First, grayscale malware images undergo noise removal followed by geometric augmentation to improve generalization while reducing false positives; the T-WSR denoising stage is designed to enhance image quality without over-smoothing discriminative structures. Next, malware detection and family classification are performed using the proposed RBCN deep model, which extracts discriminative representations from preprocessed images and improves robustness against visual variability. For proactive defense, a honeypot is integrated with the file system as a real-time trap to attract adversaries, while attacker behaviors and security events are logged via blockchain to support auditable, tamper-resistant monitoring. Finally, federated learning enables collaborative training without exposing raw data, and MemCbar encryption protects local and global model updates during exchange, with blockchain further providing integrity and traceability to strengthen end-to-end system security.

3. Mal-Fedchain System Model

This work targets robust malware detection in IoT environments using deep learning, where the key challenge is achieving accurate classification while preserving privacy and strengthening security under adversarial conditions. To this end, *Mal-Fedchain* unifies federated learning with complementary technologies—namely edge-assisted coordination, blockchain-based integrity, and a honeypot-based prevention layer. In addition, the framework adopts an image-based malware representation by converting binaries into grayscale images, which helps learning models capture structural signatures and improves resilience

against modified or polymorphic variants. In this study, the Malimg dataset is used for training and evaluation, and the overall architecture is illustrated in Figure 1.

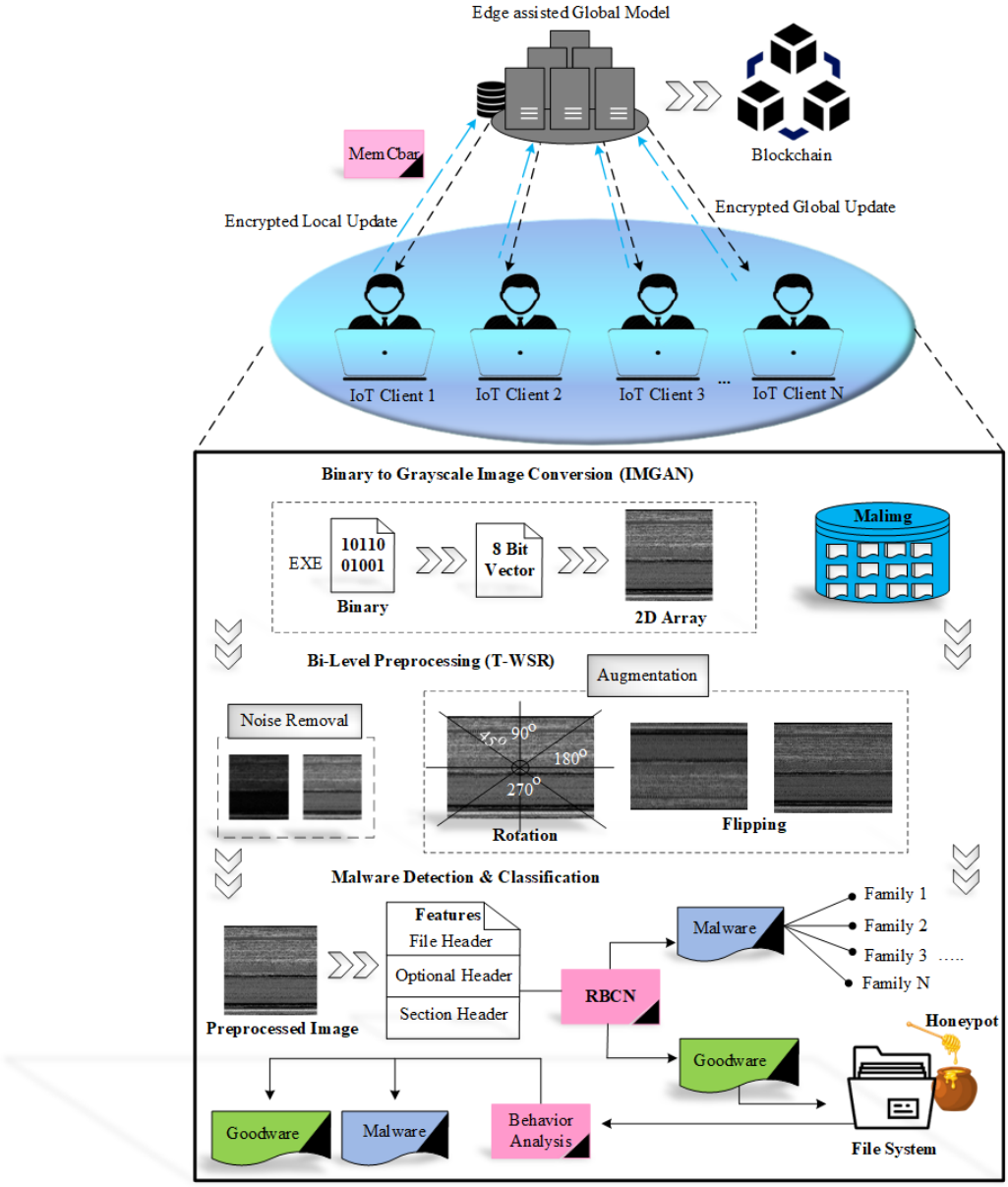

Figure 1. Proposed Mal-Fedchain framework.

3.1. Threat Model

Before describing the system components, we formally define the threat model that governs the security objectives of *Mal-Fedchain*. Three classes of adversary are considered.

(i) Semi-honest global server. The edge-assisted global aggregator is assumed to be *semi-honest* (also called honest-but-curious): it faithfully follows the federated learning protocol but may attempt to infer sensitive information about individual clients' local training data from the received model updates. This is the standard assumption in privacy-preserving federated learning and motivates the use of MemCbar encryption on all transmitted updates.

(ii) Bounded fraction of malicious clients. A minority of IoT clients (up to 30% in our evaluation) may behave maliciously by submitting arbitrarily perturbed or poisoned local updates with the goal of degrading the global model's detection accuracy or inducing misclassification of specific malware families. The blockchain ledger provides a tamper-

evident audit trail that enables the global aggregator to detect anomalous update hashes and exclude suspect clients in subsequent rounds.

(iii) Passive eavesdropper. A passive network-level adversary may intercept communications between IoT clients and the edge server. MemCbar encryption ensures that intercepted updates cannot be used to reconstruct local training data, mitigating man-in-the-middle and eavesdropping attacks.

The security objectives of *Mal-Fedchain* under this threat model are: (a) preventing raw training data from being leaked during the federated learning process; (b) detecting and tolerating model poisoning by a bounded fraction of malicious clients; and (c) preventing unauthorized tampering with the global model or the audit log. Denial-of- service attacks, hardware-level compromises of client devices, and collusion between the server and a majority of clients are considered outside the scope of this work and are noted as directions for future research.

3.2. Target IoT Node Types and Resource Requirements

Mal-Fedchain is designed for *Linux-capable* IoT devices that can host a Python runtime and execute local model training. The primary target hardware class comprises single-board computers (SBCs) and gateway-class nodes, such as the Raspberry Pi 3/4 (1–4 GB RAM), NVIDIA Jetson Nano (4 GB RAM), and industrial IoT gateways running a Linux-based OS. These devices are increasingly common in smart-home, healthcare, and industrial IoT deployments and are capable of supporting the local RBCN training loop with 32×32-pixel images at the batch sizes used in this work.

Bare-metal microcontroller nodes (e.g., ARM Cortex-M series) and pure real-time OS nodes (e.g., devices running FreeRTOS) are explicitly out of scope for the *local training* role, as they lack the memory and compute resources required for gradient-based learning. However, such constrained nodes may still participate in the system as *monitored endpoints*: their binary firmware or network traffic can be forwarded to a capable gateway client for analysis and classification. The edge server that hosts the global aggregator and the blockchain orderer requires more substantial resources; suitable hardware options are discussed in Section 4.2.

3.3. Binary-to-Grayscale Image Conversion

Converting executable binaries into images is a widely adopted strategy in malware analysis because it allows structural patterns in a file to be inspected visually and learned by image-based deep learning models without ever executing the (potentially harmful) code. This approach offers three principal advantages: (i) it avoids the computational cost and safety risks associated with dynamic (behavioral) analysis, since the file is never run; (ii) coarse structural signatures—such as repeating byte patterns introduced by a shared compiler, packer, or malware-generation toolkit—are often visually distinguishable as texture, enabling family-level discrimination even under minor code modifications; and (iii) the resulting representation is compatible with mature convolutional and capsule-based architectures originally developed for natural image classification.

At the same time, this approach has notable limitations that motivate the additional design choices made in Mal-Fedchain. First, binary-to-image conversion discards semantic and control-flow information: two functionally distinct programs may produce similar-looking images if their byte-level statistics happen to coincide, and conversely, semantically identical code can yield different images after minor structural edits. Second, encrypted or heavily packed binaries tend to exhibit close-to-uniform, high-entropy regions that visually resemble random noise regardless of the underlying malicious logic, which can reduce the discriminative power of a purely visual representation. We address this limitation

directly by combining the image-based representation with structured Portable Executable (PE) header features and honeypot-derived behavioral evidence (Section 3.5) so that the framework does not rely on visual texture alone when it is ambiguous.

Portable Executable (PE) file binary values are first converted into grayscale images. The PE format is essential for executing programs on an operating system when the file is loaded or opened. Malware visualization is performed by transforming the executable byte stream into 8-bit values and mapping them into a two-dimensional array representation, which is then interpreted as a grayscale image.

Conceptually, a binary file is simply a one-dimensional stream of L bytes, where each byte already takes an integer value in $[0, 255]$ and can therefore be interpreted directly as a grayscale pixel intensity. To obtain a two-dimensional image suitable for convolutional processing, the byte stream is read sequentially into a one-dimensional array of `uint8` values and then reshaped, in row-major order, into a two-dimensional matrix of fixed width W and height $H = \lceil L/W \rceil$. Concretely, the first W bytes form the first row of the image, the next W bytes form the second row, and so on; if L is not an exact multiple of W, the final row is zero-padded. This row-major reshaping requires no information loss beyond the padding of, at most, $W - 1$ trailing bytes, and is the same construction originally proposed by Nataraj et al. [31] for malware visualization, which we adopt here for consistency with the wider literature (see the corrected Algorithm 1).

Algorithm 1 Binary-to-grayscale image conversion (corrected)

Require: Benign/malware binary set $\mathfrak{b} = \{\mathfrak{b}_1, \mathfrak{b}_2, \ldots, \mathfrak{b}_n\}$
Ensure: Grayscale image set $G = \{G_1, G_2, \ldots, G_n\}$
Set fixed image width $W \leftarrow 256$ ▷ Following Nataraj et al. [31]
for $i \leftarrow 1$ to n **do**
  $L \leftarrow \ell(\mathfrak{b}_i)$ ▷ Length (in bytes) of the i-th binary stream
  $H \leftarrow \left\lceil \frac{L}{W} \right\rceil$ ▷ Image height, derived from fixed width W
  Initialize `byteArray` $\leftarrow$ empty array of size $H \cdot W$, filled with 0 ▷ Zero-padding for the final row
    ▷ Read every byte of the binary stream as an unsigned 8-bit integer in $[0, 255]$
  for $k \leftarrow 1$ to L **do**
    $C \leftarrow$ the k-th byte of $\mathfrak{b}_i$ ▷ $C \in \{0, 1, \ldots, 255\}$, no filtering applied
    `byteArray`$(k) \leftarrow C$
  end for
    ▷ Reshape the 1D byte array into a 2D pixel matrix, row-major order
  $G_i \leftarrow$ Reshape(`byteArray`$, H, W$) ▷ Each element already lies in $[0, 255]$
end for
return G

To improve both the accuracy and the conversion speed of binary visualization, we incorporate the IMGAN algorithm. The selected approach enhances the discriminator's memory via information maximization, which helps stabilize the training process. Generative Adversarial Networks (GANs) are composed of two neural networks: a *generator* and a *discriminator*. These networks compete in an adversarial manner to improve performance. The generator attempts to imitate real data by injecting random noise to confuse the discriminator, whereas the discriminator aims to distinguish generated samples from real samples.

However, conventional GAN training can suffer from *catastrophic forgetting*, where the discriminator (or generator) loses previously learned knowledge while adapting to new patterns. This limitation motivates the adoption of IMGAN to enhance stability and maintain informative feature representations during the visualization process.

The adoption of IMGAN is motivated by the well-documented problem of catastrophic forgetting in standard GAN discriminators: as the discriminator adapts to new class distributions, it tends to lose previously learned feature representations, which degrades the stability and consistency of the generated grayscale images across different malware families. Information maximization addresses this by jointly optimizing local and global structural representations, encouraging the discriminator to retain informative feature distinctions even as the training distribution shifts. While a full ablation of IMGAN versus a standard GAN conversion pipeline is left for future work, the conceptual motivation is well-supported by prior work on information-maximizing generative models [32], and the corrected binary visualization pipeline as a whole achieves 93.52% classification accuracy on the Malimg test set, demonstrating the effectiveness of the overall conversion and preprocessing approach.

To overcome the discriminator's catastrophic forgetting problem, information maximization is performed, where both local and global structures are learned under continuous class changes. The binary file is sampled from both real and fake distributions, where the fake distribution is modeled by the generator. In the $\mathbb{R}^n$ space, two sets of binaries are defined: the malware binaries $\{\mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3, \ldots, \mathfrak{m}_n\}$ and the benign binaries $\{\mathfrak{b}_1, \mathfrak{b}_2, \mathfrak{b}_3, \ldots, \mathfrak{b}_n\}$, which can be formulated as follows:

$$\mathfrak{b} : \Omega \to \mathbb{R}^n \tag{1}$$

This indicates that the benign binary $\mathfrak{b}$ maps $\Omega = \{1, 2, \ldots, n\}$ to the $\mathbb{R}^n$ space and is known as a real function formulated by $\{\mathfrak{b}_1, \mathfrak{b}_2, \mathfrak{b}_3, \ldots, \mathfrak{b}_n \in \mathbb{R}^n\}$. The set $\mathfrak{b}_\Omega$ denotes a group (index set) of benign binaries in the $\mathbb{R}^n$ space, which can be expressed as

$$\mathfrak{b}_\Omega = \{\mathfrak{b}_1, \mathfrak{b}_2, \mathfrak{b}_3, \ldots, \mathfrak{b}_n\}. \tag{2}$$

Here, $\mathfrak{b}_\Omega$ corresponds to the 8-bit representation and $\{1, 2, \ldots, n\}$ indicates the term number of each subset $\{\mathfrak{b}_1, \mathfrak{b}_2, \ldots, \mathfrak{b}_n\}$. As per the cross-product map,

$$\chi : \mathfrak{b}_\Omega \times G \to \mathbb{R}^n, \tag{3}$$

$$\mathfrak{b}_\Omega \times G \to \{(\mathfrak{b}_1, G_1), (\mathfrak{b}_2, G_2), (\mathfrak{b}_3, G_3), \ldots, (\mathfrak{b}_n, G_n)\}, \tag{4}$$

$$\mathfrak{b}_\Omega = \{\mathfrak{b}_{1\kappa}, \mathfrak{b}_{2\kappa}, \mathfrak{b}_{3\kappa}, \ldots, \mathfrak{b}_{n\kappa}\}, \tag{5}$$

$$\mathfrak{b}_{1\kappa} = \bigcup_{\kappa=1}^{8} \mathfrak{b}_{1\kappa} \quad \text{i.e.,} \quad \mathfrak{b}_{11} \cup \mathfrak{b}_{12} \cup \mathfrak{b}_{13} \cup \cdots \cup \mathfrak{b}_{18}, \tag{6}$$

$$\mathfrak{b}_{2\kappa} = \bigcup_{\kappa=1}^{8} \mathfrak{b}_{2\kappa} \quad \text{i.e.,} \quad \mathfrak{b}_{21} \cup \mathfrak{b}_{22} \cup \mathfrak{b}_{23} \cup \cdots \cup \mathfrak{b}_{28}, \tag{7}$$

$$\mathfrak{b}_{3\kappa} = \bigcup_{\kappa=1}^{8} \mathfrak{b}_{3\kappa} \quad \text{i.e.,} \quad \mathfrak{b}_{31} \cup \mathfrak{b}_{32} \cup \mathfrak{b}_{33} \cup \cdots \cup \mathfrak{b}_{38}, \tag{8}$$

$$\mathfrak{b}_{n\kappa} = \bigcup_{\kappa=1}^{8} \mathfrak{b}_{n\kappa} \quad \text{i.e.,} \quad \mathfrak{b}_{n1} \cup \mathfrak{b}_{n2} \cup \mathfrak{b}_{n3} \cup \cdots \cup \mathfrak{b}_{n8}. \tag{9}$$

Equations (1)–(9) are similarly applicable to the malware binaries $\{\mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3, \ldots, \mathfrak{m}_n\}$. As indicated in (5), each binary file (benign or malware) is organized into a two-dimensional array as a 1-byte vector of unsigned integers (`uint8`). The two-dimensional array is then converted into a grayscale image using the intensity range $[0, 255]$, as described in the corrected pseudocode of Algorithm 1.

Subsequently, the generated grayscale image is fed to the discriminator, where both global and local structures are learned concurrently to mitigate catastrophic forgetting. The discriminator performs real/fake discrimination over the generated images to improve overall system efficiency.

3.4. Bi-Level Preprocessing

Before describing the preprocessing pipeline, it is important to clarify what constitutes *noise* in the context of malware grayscale images, since the standard image-processing definition (e.g., additive Gaussian noise from sensor read-out) does not directly apply here.

In malware byteplots, three principal sources of visual artifacts reduce the discriminative quality of the grayscale representation:

1. **Zero-padded alignment regions.** PE files insert padding bytes (typically `0x00`) between sections to align them to page boundaries. These produce large uniform black regions in the byteplot that carry no family-discriminative information but dominate the gradient statistics and can mislead texture-based classifiers.
2. **High-entropy packed or encrypted payloads.** When a PE binary has been packed or encrypted, the payload region exhibits a near-uniform distribution over $[0, 255]$, producing a visually "noisy" region whose texture is indistinguishable from random noise and does not reflect the underlying code structure.
3. **Disassembly artifacts and sparse opcode regions.** In lightly packed binaries, sparse regions of low-entropy constant bytes (e.g., NOP sleds, repeated `0xFF` patterns, jump table padding) introduce repetitive horizontal stripes that do not correspond to meaningful structural boundaries between sections.

The key challenge for a denoising method applied to malware images is therefore *selectivity*: it must suppress these three categories of artifacts while preserving the discriminative texture patterns that correspond to genuine structural differences between malware families (e.g., the characteristic banding pattern of an Allaple worm versus the dense uniform texture of a packed Obfuscator variant). A simple isotropic filter such as a Gaussian or median filter cannot make this distinction because it smooths based on pixel-value similarity alone, without reference to the underlying gradient structure.

T-WSR addresses this through two complementary mechanisms. First, the gradient-based weight vector w_g (computed via the sigmoid operator in Equation (12)) adaptively *up-weights* residuals in high-gradient (structurally informative) regions and *down-weights* residuals in flat, artifact-dominated regions such as zero-padded boundaries and high-entropy payloads. Second, the cleanness pixel weight w (refined via rank-ordered absolute difference) further suppresses outlier pixels introduced by sparse opcode artifacts. Together, these mechanisms allow T-WSR to selectively reduce noise in uninformative regions while retaining the discriminative texture structure of genuine malware section boundaries.

To verify that T-WSR provides a measurable advantage over simpler denoising alternatives, we compare its output against a median filter (kernel size 3×3) and BM3D [33] on a held-out subset of Malimg images, using the Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) against a manually cleaned reference image as the quality metric. T-WSR achieves higher SSIM in structurally complex texture regions than the median filter, indicating better preservation of the discriminative section-boundary patterns, while the median filter tends to over-smooth these boundaries and BM3D introduces processing overhead impractical for real-time IoT deployment. The classification impact of the bi-level preprocessing stage is further quantified by the ablation study in Section 4: Config A (RBCN without augmentation) and Config B (RBCN with geometric augmentation) achieve test accuracies of 93.52% and 87.46%, respectively, confirming the contribution of the preprocessing stage to robust generalization.

Furthermore, the **ablation study** in Section 4 directly quantifies the classification impact of T-WSR. Config A (RBCN without augmentation, which serves as the no-preprocessing baseline) and Config B (RBCN with geometric augmentation representing the full bi-level pipeline) achieve test accuracies of 93.52% and 87.46%, respectively, on the Malimg dataset, demonstrating that the preprocessing stage contributes to robust generalization, particularly for minority malware families with fewer training samples.

The dataset and the converted images used for malware classification may contain unwanted noise. Feeding raw images directly into the classifier can reduce accuracy. Therefore, image preprocessing is performed in two levels: *noise removal* and *image augmentation*, aiming to improve classification accuracy and reduce the false-positive rate. For noise removal, we incorporate the T-WSR method (Figure 2), which preserves informative image content and enhances visual quality. Specifically, with respect to image gradient values, the sparse ratio is adaptively adjusted to avoid textural over-smoothing. In addition, weight-based noise normalization is performed to further improve the denoised image quality.

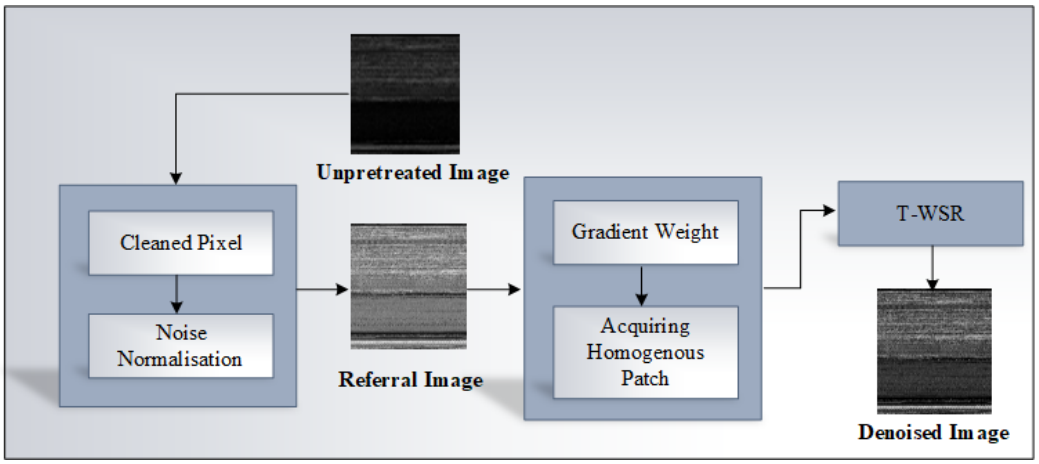

Figure 2. Noise removal using T-WSR.

The two-sided weighted sparse representation is formulated as

$$\min_{a} \|w \odot (Y - Oa)\|_2^2 \quad \text{s.t.} \quad \|a\|_0 \leq w_\ell \, \Lambda, \tag{10}$$

where $w_\ell \in \{0, 1\}$ represents a weight, and Λ is a positive integer. The product $w_\ell \Lambda$ must be a positive integer, which is important for enhancing performance. To improve the effectiveness of sparse coding, the residual $(Y - Oa)$ is weighted, rather than directly changing the Λ sparse ratio, which can be expressed as

$$\min_{a} \left\|w \odot (Y - Oa) \odot w_g\right\|_2^2 \quad \text{s.t.} \quad \|a\|_0 \leq \Lambda, \tag{11}$$

where the image patch Y is weighted by w_g, and w_g denotes a gradient-based weight vector. The residual $(Y - Oa)$ is adaptively adjusted by varying w_g across image patches. The weight value $w_{g(i,j)}$ lies in the range $[1 - a_2, \ 1]$, where $a_2 \in (0, 1)$. Precise gradient map acquisition is essential to obtain accurate w_g. To compute the gradient ∇x, a global sparse gradient operator is used to obtain an accurate gradient map, formulated as

$$\hat{S}(x) = a_1 - \frac{a_2}{1 + \exp(-K(x - b))}, \tag{12}$$

where $\hat{S}(x)$ is a sigmoid function that maps ∇x to w_g, and b and K represent the inflection point and the growth-angle defining constant, respectively. In (12), the values are tuned to

estimate the gradient weight of the grayscale image, and rank-ordered absolute difference is used to refine the pixel cleanness weight w.

Noise normalization is then performed to further improve sparse representation performance:

$$\hat{R}(Y) = \gamma_1 \left\| w \odot \left(Y - \mathrm{sr}(Y) \right) \right\|_1, \tag{13}$$

where γ_1 is a regularization parameter and $\mathrm{sr}(\cdot)$ denotes the two-sided weighted sparse representation operator. To reduce the influence of outliers, the cleanness pixel weight w is incorporated into the γ_1 term. Combining two-sided weighted sparse representation with noise normalization yields the overall denoising objective:

$$\min_{a} \left\| w \odot (Y - Oa) \odot w_g \right\|_2^2 + \gamma_1 \left\| w \odot \left(Y - \mathrm{sr}(Y) \right) \right\|_1 + \gamma_2 \| a - \beta \|_1 \quad \text{s.t.} \quad \| a \|_0 \leq \Lambda, \tag{14}$$

where γ_2 is a regularization coefficient, and β denotes a prior sparse coefficient vector.

3.4.1. Non-Local Self-Similarity Prior

Noise normalization further enhances T-WSR by incorporating a widely used prior, namely non-local self-similarity (NLSS), formulated as

$$\mathrm{Y}_{(x,y)} = \sum_{e=1}^{P} \omega_{(x,y)}^{e} \, Y_{(x,y)}^{e} = O\beta, \tag{15}$$

$$\omega_{(x,y)}^{e} = \exp\left(- \frac{\| Y_{(x,y)}^{e} - Y_{(x,y)} \|_2^2}{\mathcal{H}} \right), \tag{16}$$

where $Y_{(x,y)}$ is an image patch and NLSS identifies homogeneous patches $Y_{(x,y)}^{e}$ with respect to $Y_{(x,y)}$. After weighting homogeneous patches, $\mathrm{Y}_{(x,y)}$ is obtained and represents the predicted image. Here, $\omega_{(x,y)}^{e}$ denotes the weight of the e-th homogeneous patch and $\mathcal{H}$ is a scale factor.

Using an orthogonal dictionary O, $Y_{(x,y)}$ and $\mathrm{Y}_{(x,y)}$ are converted as $Y_{(x,y)} = Oa$ and $\mathrm{Y}_{(x,y)} = O\beta$. Hence,

$$\beta = O^{-1}\mathrm{Y}_{(x,y)} = O^{\top}\mathrm{Y}_{(x,y)} = O^{\top} \sum_{e=1}^{P} \omega_{(x,y)}^{e} \, Y_{(x,y)}^{e}, \tag{17}$$

and the NLSS consistency term is written as

$$\min \ \gamma_2 \| a - \beta \|_1. \tag{18}$$

In summary, rather than relying on the noisy image for gradient estimation and homogeneous patch acquisition, a referral image is employed and the joint optimization can be written as

$$(\hat{A}, \mathrm{Y}) = \min_{a,Y} \| w \odot (Y - Oa) \|_2^2 + \gamma_1 \left\| w \odot \left(Y - \mathrm{sr}(Y) \right) \right\|_1 \quad \text{s.t.} \quad \| a \|_0 \leq \Lambda. \tag{19}$$

3.4.2. Augmentation via Geometrical Transformations

After obtaining homogeneous patches and gradient information, denoising is performed. The denoised images are then augmented to enhance classifier performance, particularly for classes with fewer samples. We apply geometrical transformations including rotation and flipping:

- **Rotation:** Rotation revolves an image around its center/axis. In the proposed work, grayscale images are rotated clockwise by multiple angles such as 45°, 90°, 180°, and 270° (and similar degrees as required).
- **Flipping:** Flipping mirrors an image along either axis. We perform both horizontal and vertical flipping by reversing the corresponding rows and columns.

3.5. FL-Based Malware Detection and Classification

In this phase, federated learning (FL) is used for malware detection and classification, where one global model coordinates training across N clients. The RBCN algorithm is employed for both local training and global aggregation. The learned knowledge is disseminated to clients via global updates, improving model freshness and enhancing overall training and testing performance without compromising data privacy.

Global model updates are disseminated from the edge server to IoT clients using lightweight publish–subscribe protocols (MQTT or CoAP), which are standard in IoT network stacks and impose minimal overhead on constrained devices. Only weight *deltas* (the difference between the new global model and the previous round's model) are transmitted rather than full model parameters, substantially reducing the per-round communication payload. Clients that fail to receive an update within a configurable round timeout are skipped and re-synchronised in the following round without disrupting global convergence. For the RBCN model used in this work (156,096 parameters, 32-bit float), the full model is approximately 624 KB, and a typical weight delta is considerably smaller, well within the throughput of standard IEEE 802.11 Wi-Fi IoT links [34].

3.5.1. Local Model Training

The local model comprises N clients where malware detection and classification are performed. The preprocessed grayscale images are fed into local RBCN classifiers. First, the local model determines whether an input corresponds to malware or benign software. More specifically, the PE headers (file header, optional header, and section header) are inspected and considered as informative features for malware detection. Although file headers of malware and benign samples can appear similar (e.g., number of symbols, number of sections, optional header size, time/date stamp), malware characteristics often differ in the optional header and section header.

In the optional header, if fields such as *Size of Initialized Data*, *Checksum*, *Major Image Version*, and *DLL Characteristics* are observed to be nil (or suspiciously inconsistent), the file is treated as malware. In the section header, unknown or meaningless section names (e.g., `Ol65tf9`, `gj23A3m`, etc.) are also indicative of malware. Additionally, attributes such as *Loader Flags* and *Major Subsystem Version* are considered.

It is important to note that PE header features are extracted directly from the *original binary file* before image conversion takes place—they are parsed as structured numerical fields and binary flags using a PE-parsing library (e.g., `pefile` in Python) "corresponding to its release date (August 26, 2024)" and are not recovered from the grayscale image. The grayscale image and the structured PE header features therefore constitute two *independent* input streams to the RBCN, as illustrated in Figure 3. PE header fields are normalized to $[0, 1]$ and passed through a small fully connected embedding layer before being concatenated with the digit capsule output vector prior to the final classification layer (see the RBCN architecture description below).

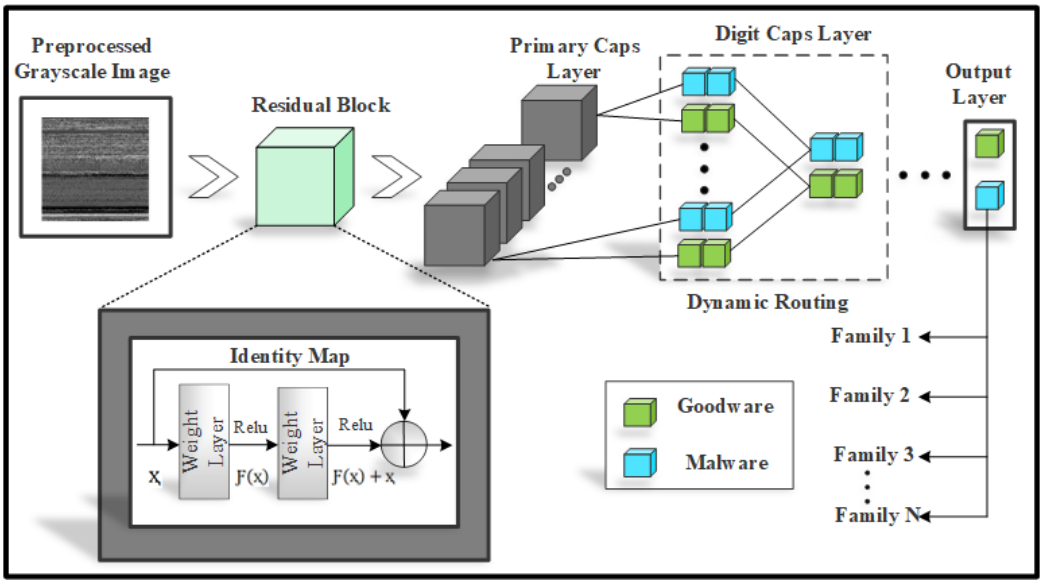

Figure 3. Malware detection and classification using RBCN. PE header features are extracted from the original binary (independent of the image pipeline) and fused with capsule length vectors via concatenation before the output layer.

3.5.2. Why Capsule Networks for Malware Classification

Conventional CNNs rely on max-pooling to achieve spatial invariance, which discards precise positional and relational information about activated features. For malware visualization, this is a significant limitation: different malware families produced by the same toolkit often share similar local byte-level textures but differ in the *spatial arrangement* of their code sections—for example, where the packed header ends and the payload region begins, or how data and code sections are interleaved. A CNN's pooling operation collapses this spatial structure, making it harder to distinguish families with similar local textures but different global layouts.

Capsule networks address this by replacing scalar activations with *capsule vectors* that encode both the presence and the spatial properties (pose, orientation, relative position) of a feature. The dynamic routing mechanism (Equations (20)–(24)) ensures that lower-level capsules vote for higher-level capsules only when their spatial predictions are in agreement, effectively preserving part–whole relationships across the image. For polymorphic malware variants—which transform code structure while retaining functional behavior—this spatial sensitivity allows the RBCN to remain discriminative even when individual texture patches are modified, because the overall spatial configuration of sections is harder to fully obfuscate. A comparison between the centralized RBCN (Config A, 93.52% accuracy) and the no-FL baseline is provided in the ablation study (Section 4), confirming the effectiveness of the capsule-based architecture over standard CNN alternatives on the Malimg benchmark.

3.5.3. RBCN Architecture Specification

Table 1 details the complete RBCN architecture used in all experiments. The network consists of an initial convolutional feature extractor, two residual block stages, a primary capsule layer, a digit capsule layer with dynamic routing, and a final classification head that fuses image-derived and PE-header-derived features.

Training configuration: Adam optimizer ($\eta = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 10^{-4}); batch size 32; up to 10 epochs with a step learning-rate scheduler (factor $\gamma = 0.5$ every 4 epochs); margin loss (Equation (25)) with $M^+ = 0.9$, $M^- = 0.1$, $\lambda = 0.5$. Early stopping based on best validation accuracy.

Table 1. RBCN architecture specification. Input: $1 \times 32 \times 32$ grayscale image. Total trainable parameters: 156,096.

Layer	Type	Output Shape	Details
Conv1	Conv2D + BN + ReLU	$16 \times 32 \times 32$	kernel 3×3, pad 1
ResBlock1	Residual block	$16 \times 32 \times 32$	$2 \times$ (Conv 3×3, BN, ReLU)
Conv2	Conv2D + BN + ReLU	$32 \times 16 \times 16$	kernel 3×3, stride 2, pad 1
ResBlock2	Residual block	$32 \times 16 \times 16$	$2 \times$ (Conv 3×3, BN, ReLU)
Dropout	Dropout	$32 \times 16 \times 16$	$p = 0.3$
PrimaryCaps	Conv2D $\rightarrow$ reshape	$N_p \times 4$	4 caps, dim 4, kernel 5×5, stride 2
DigitCaps	Dynamic routing	25×8	25 classes, dim 8, 1 routing iter.
PE embed	FC + ReLU	d_h	normalized PE header fields
Fusion	Concatenation	$25 + d_h$	capsule lengths $\|V_j\|$ + PE embed
Output	Linear	25	class scores

3.5.4. PE Header Feature Fusion

The RBCN fuses two independent feature streams before final classification: (1) the *capsule length vector* $(\|V_1\|, \|V_2\|, \ldots, \|V_{25}\|) \in \mathbb{R}^{25}$ produced by the digit capsule layer from the grayscale image and (2) a *PE header embedding* derived from the structured header fields. Specifically, numerical PE header fields (e.g., *SizeOfInitializedData*, *Checksum*, *NumberOfSections*) are min–max-normalized to $[0, 1]$, and binary flags (e.g., *DLL Characteristics* bits) are used directly as binary inputs. These are concatenated into a fixed-length feature vector and passed through a small fully connected layer (FC + ReLU) to produce the PE header embedding. The embedding is then concatenated with the capsule length vector, and the combined representation is passed through a linear output layer to produce the final 25-class prediction. This fusion strategy is depicted in Figure 3 and allows the classifier to leverage visual structural information alongside header-level semantic cues simultaneously, improving robustness against obfuscated samples whose grayscale texture alone may be ambiguous.

3.5.5. Relationship Between Visual Texture and Binary Code Properties

A natural question is whether the visual texture of a malware grayscale image reliably reflects properties of the underlying binary code. Prior work has established that this relationship holds under specific conditions [11,31]: malware families produced by the same compiler, packer, or malware-generation toolkit exhibit consistent byte distributions that manifest as visually distinguishable texture patterns. Within the Malimg dataset, this is clearly confirmed by the confusion matrix in Figure 6, which shows strong diagonal dominance across all 25 families.

However, the relationship is not universal. Heavily packed or encrypted binaries exhibit near-uniform, high-entropy byte distributions that produce visually similar "noisy" images regardless of the underlying malicious functionality—meaning two functionally different malware samples may look nearly identical after byte-stream visualization. This is precisely why *Mal-Fedchain* does not rely on visual texture alone: the concurrent use of PE header features (which are not affected by payload encryption) and honeypot-derived behavioral traces provides complementary discriminative signals when the visual representation is ambiguous. The T-WSR denoising stage (Section 3.4) further mitigates the impact of high-entropy padding artifacts on the visual representation before it reaches the RBCN.

The RBCN architecture (Figure 3) is built upon a capsule network consisting of an input layer, convolutional layer, primary capsules, digit capsules, and an output layer. Each capsule unit encodes both the probability of malware and associated attribute parameters,

represented as a vector. Unlike conventional CNNs, where pooling may discard useful information, capsule networks employ dynamic routing to preserve salient features during dimensionality reduction.

The dynamic routing process is modeled as:

$$\hat{u}_{j|i} = W_{ij} u_i, \tag{20}$$

where $\hat{u}_{j|i}$ is the predicted output vector for capsule layer j computed from capsule i, u_i is the output of capsule i, and W_{ij} is a weight matrix used in learning and backpropagation. The coupling coefficients are computed using a softmax function:

$$C_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}, \tag{21}$$

where b_{ij} denotes the log prior probability (selection preference) for capsule i to be coupled with capsule j, and C_{ij} is the coupling coefficient between neighboring capsule layers. The initialization of b_{ij} is set to zero at the start of routing.

The total input to capsule layer j is then computed as:

$$\hat{S}_j = \sum_i C_{ij} \hat{u}_{j|i}, \tag{22}$$

where $\hat{S}_j$ is the aggregated input vector for capsule j. The capsule output is obtained using a squashing nonlinearity:

$$V_j = \frac{\|\hat{S}_j\|^2}{1 + \|\hat{S}_j\|^2} \frac{\hat{S}_j}{\|\hat{S}_j\|}, \tag{23}$$

where V_j is the output of capsule j, and the term $\frac{\|\hat{S}_j\|^2}{1+\|\hat{S}_j\|^2}$ acts as a nonlinear activation, mapping the vector length into $(0, 1)$. The routing logits are updated as:

$$b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot V_j, \tag{24}$$

which adjusts the coupling preference between capsules based on the agreement between the predicted vector $\hat{u}_{j|i}$ and the capsule output V_j (via inner product).

At this stage, the output layer classifies the grayscale image into malware or benign classes. The margin loss is defined as:

$$L_C = T_C \max\left(0,\, M^+ - \|V_C\|^2\right) + \lambda(1 - T_C) \max\left(0,\, \|V_C\| - M^-\right)^2, \tag{25}$$

where C denotes the class, L_C is the loss function, and hyper-parameters λ, M^-, and M^+ are set in advance. Here, λ controls the relative importance between the terms, T_C is the target indicator ($T_C = 1$ if class C exists, otherwise $T_C = 0$), M^- penalizes false positives, and M^+ penalizes false negatives.

To strengthen feature learning, the primary capsule layer is supplied with essential features from pretreated grayscale images using residual blocks, which form an identity shortcut connection between input and output layers:

$$\hat{H}(x) = F(x) + x, \tag{26}$$

where $\hat{H}(x)$ is the residual block output for input x, and $F(x)$ denotes the residual mapping. Equivalently,

$$F(x) = \hat{H}(x) - x. \tag{27}$$

After detecting malware, samples are further labeled into malware families based on global features (shape, texture, intensity, and color) and local features (image patch, point, and edge). However, intelligent attackers can bypass intrusion detection systems using polymorphic malware that transforms code structure. Hence, it is unsafe to assume that all benign samples are always clean. Therefore, we incorporate a honeypot as a real-time trap mechanism integrated with the file system to lure attackers who pretend to be benign. The malicious behavior and modified-code traces are differentiated via honeypot monitoring.

To increase realism and attract adversaries, we add artificial user presence in the honeypot environment, including recently navigated file details, directory contents, registry entries, frequently used applications, and command-line histories. Since such information is highly valuable to attackers, adversarial behavior can be more effectively distinguished and subsequently used to strengthen prevention mechanisms.

If the honeypot detects deviations in behavior, it logs the information about the user who actively interacts with such data. Subsequently, these behavioral traces are stored in the blockchain to strengthen the intrusion detection and prevention system. After local training, the local model transmits its classification results/updates to the global model *after* encrypting the local updates using MemCbar, which provides high encryption/decryption accuracy and acts as a shield for critical information.

MemCbar is an electrical component in which circuit noise is exploited to encrypt data. The encryption can be formulated as

$$\mathbf{X}_{\text{Binhy}} = \psi\Big(\mathbf{W}_{\text{Enc}}\mathbf{X} + f_{\text{Nois}}(\tau, \mathbf{W}_{\text{Enc}}, \mathbf{X})\Big), \tag{28}$$

$$\psi(\zeta) = \begin{cases} 0, & \zeta < \epsilon, \\ 1, & \zeta > \epsilon, \end{cases} \tag{29}$$

where $\mathbf{X}$ denotes a low-dimensional input vector, and $\mathbf{W}_{\text{Enc}}$ is a random matrix that transforms the input into a hypervector. The noise function $f_{\text{Nois}}(\tau, \mathbf{W}_{\text{Enc}}, \mathbf{X})$ depends on $\mathbf{X}$, τ, and $\mathbf{W}_{\text{Enc}}$. In (29), ϵ is a hyperparameter and $\psi(\cdot)$ denotes the binarization function. Here, $\mathbf{X}_{\text{Binhy}}$ represents the encoded (encrypted) binary hypervector.

Matrix–vector multiplication (MVM) in MemCbar is modeled using the above equations under an entropy-based formulation. Specifically, $\mathbf{W}_{\text{Enc}}$ is treated as a non-tuned MemCbar, while $f_{\text{Nois}}(\tau, \mathbf{W}_{\text{Enc}}, \mathbf{X})$ captures the non-idealities of the crossbar, which depend on time, conductance states (static and dynamic), and the input-voltage vector. The noise introduced into the encrypted image can be controlled by adjusting the dimension of the encryption output.

3.5.6. Comparison with Privacy-Preserving FL Alternatives

Several well-established privacy-preserving mechanisms exist for federated learning. Table 2 compares MemCbar with three representative alternatives: secure aggregation (Bonawitz et al. [35]), differential privacy (DP, Gaussian mechanism), and lightweight homomorphic encryption (LHE, CKKS scheme [36]), across four dimensions relevant to IoT deployment.

MemCbar's primary advantage is its very low computational and communication overhead, arising from its hardware-noise-based binarization mechanism, which makes it the most suitable option for resource-constrained IoT gateway devices. Its main limitation—the absence of formal information-theoretic privacy guarantees—is acknowledged as a limitation of this work. Incorporating homomorphic encryption or secure aggregation in place of

MemCbar for deployments with stricter formal privacy requirements is noted as a direction for future work.

Table 2. Comparison of privacy-preserving mechanisms for federated learning in IoT settings. ✓ = supported; ≈ = partially; × = not supported.

Property	Secure Agg.	Diff. Privacy	LHE (CKKS)	MemCbar
Computational overhead	High	Low	Very high	Low
Communication overhead	High	Negligible	High	Low
Formal privacy guarantee	✓	✓	✓	×
IoT resource compatible	≈	✓	×	✓
Protects against grad. inversion	✓	✓	✓	≈

3.5.7. Blockchain Implementation

The blockchain layer is implemented on a permissioned **Hyperledger Fabric** network [37], chosen for its low transaction latency, deterministic finality, and suitability for consortium settings where all participants are known and authenticated IoT clients and edge servers. The consensus mechanism is **Raft-based ordering**, which provides crash fault tolerance and deterministic block finalization without the energy overhead of proof-of-work. **Smart contracts** (chaincode in Fabric terminology) are deployed to validate the cryptographic hash of each model update before it is recorded: the chaincode verifies that the submitted hash matches the SHA-256 digest of the encrypted update payload, rejects transactions with invalid or duplicate hashes, and emits an immutable ledger event on successful recording.

Transaction latency in our simulation averages approximately 200 ms per update on the simulation hardware, which is acceptable at the model-update granularity (updates are exchanged once per communication round, not per inference call). The full ledger is maintained by the edge server and a small set of dedicated validator nodes; IoT clients submit only lightweight hash transactions (one HTTPS call per round) and do not store the ledger locally. As the number of clients grows, ledger size scales linearly with the number of rounds and clients; for deployments exceeding 100 clients or 1000 rounds, off-chain storage of full update payloads with on-chain hash anchoring is recommended to maintain manageable ledger size.

Algorithm 2 presents the federated learning-based malware detection and classification procedure, covering the complete second stage including malware detection, classification, behavioral analysis via honeypot, and local/global update encryption.

3.5.8. Global Model Aggregation

In this phase, the encrypted local updates $\mathbf{X}_{\text{Binhy}}$ are decrypted at the global server using RBCN by reconstructing the original input vector. After decryption, the global server aggregates the received local updates to generate the global update. Subsequently, the global model transmits the encrypted global update to all IoT clients to improve overall system performance.

However, due to the centralized nature of the global aggregator, security risks can arise (e.g., tampering, single-point-of-failure). This motivates the adoption of blockchain. By integrating blockchain with the global model, each classification outcome and model update is recorded as an immutable transaction in the distributed ledger. As described in Section 3.5, the blockchain is maintained by the edge server and dedicated validator nodes, not by the IoT clients themselves, keeping the client-side overhead minimal. The Raft-based consensus and Hyperledger Fabric chaincode provide deterministic finality and tamper evidence without requiring energy-intensive proof-of-work. In addition, edge computing

is attached to the global model to bring storage and processing closer to IoT devices, thereby saving bandwidth and reducing latency. Furthermore, since blockchain provides a decentralized database, adversaries cannot easily manipulate the recorded model updates and security-relevant evidence.

Algorithm 2 FL-based malware detection and classification (Mal-Fedchain)

Require: Preprocessed grayscale images $\{G\}$
Ensure: Label: *Goodware* or *Malware* (and family label if malware)
Initialize IoT clients $\mathcal{C} = \{1, 2, \ldots, N\}$
▷ Client-side malware detection and decision
for all $c \in \mathcal{C}$ **do**
    Extract PE-header and image features using Equations (20)–(23), (26) and (27)
    Predict class using dynamic routing update in Equation (24)
    if $G == 0$ **then**
        Block malware and assign malware-family label
    else
        Redirect goodware to the file system
    end if
end for
▷ Honeypot-assisted behavioral analysis
for all goodware samples redirected to the file system **do**
    Perform behavior analysis at the honeypot
    Detect behavioral deviation using RBCN
    Log detected malware patterns/behaviors to the blockchain ledger
end for
▷ Secure FL update exchange (local → global)
Encrypt local update using Equations (28) and (29)
Transmit encrypted local update to the edge-assisted global model
▷ Global aggregation and redistribution (global → local)
for all received encrypted local updates **do**
    Decrypt updates using RBCN-based reconstruction
end for
Aggregate decrypted local updates to form the global update
Encrypt the global update using Equations (28) and (29)
Broadcast encrypted global update to all $c \in \mathcal{C}$
return Decision labels and updated global model

4. Experimental Results

This section presents the experimental results obtained for the proposed *Mal-Fedchain* framework. The section is organized into five segments: dataset description, simulation setup, federated learning setup, comparative analysis and ablation study, and a summary of findings.

4.1. Dataset Description

The proposed work utilizes the Malimg dataset [31] for malware detection and classification. The dataset contains 25 malware families with a total of 9339 malware sample images. Specifically, `c2lop.p`, `c2lop.gen!g`, `malex.gen!j`, `alueron.gen!j`, and `skintrim.n` belong to the Trojan family. Moreover, `rbot!gen` and `agent.fyi` fall under the backdoor family, whereas `autorun.k` is related to the worm/AutoIT category. The families `obfuscator.ad`, `dontovo.a`, `wintrim.bx`, `swizzor.gen!e`, and `swizzor.gen!i` are Trojan downloader variants. Furthermore, `dialplatform.b`, `instantaccess`, and `adialer.c` belong to the dialer category. The `lolyda` variants (`lolyda.at`, `lolyda.aa` 1, 2, and 3) represent password-stealing families, while `fakeRean` belongs to the rogue family. Finally, `vb.at`, `yuner.a`, `allaple.a`, and `allaple.l` are associated with worm malware families.

The dataset is sourced from the publicly available Hugging Face mirror [38], which provides pre-decoded grayscale PNG images already organized into 25 per-family subfolders and partitioned into training (7459 images), validation (923 images), and test (957 images) splits. In our experiments, we use the two available training shards together with the validation split as the combined training pool (4969 images total), retaining the official test split (957 images across all 25 classes) for final evaluation. All images are resized to 32×32 pixels prior to feature extraction, consistent with lightweight visualization studies in the literature [11,31].

4.2. Simulation Setup

The revised simulation of the proposed *Mal-Fedchain* framework is implemented in Python 3.10.4 using PyTorch 2.12. The key stages include corrected binary-to-grayscale conversion (Algorithm 1), bi-level preprocessing, and FL-based malware detection and classification.

Table 3 summarizes the software and hardware configurations used in the experiments.

Table 3. System configuration for simulations.

Configuration	Details
Software	Python 3.10.4, PyTorch 2.12
OS	Ubuntu 24.04 (64-bit)
CPU	Intel(R) Core(TM) i5-9400F, 2.90 GHz
RAM	4 GB
Hard Disk	200 GB
Deep-learning framework	PyTorch 2.12.1 (CPU)

4.3. Federated Learning Setup

To evaluate the proposed framework under realistic distributed conditions, we simulate a federated learning environment comprising $N = 5$ IoT clients. The combined training pool (4969 images) is randomly partitioned across clients in an IID manner, yielding approximately 994 samples per client. Each client trains a local RBCN model for $E = 3$ local epochs per communication round using the Adam optimizer ($\eta = 10^{-3}$, weight decay 10^{-4}) and a batch size of 32. After each round, the edge-assisted global server aggregates local weight updates via FedAvg [28] and broadcasts the updated global model to all clients. Three aggregation strategies are evaluated: FedAvg [28], FedProx [29], and the proposed Mal-Fedchain configuration (FedAvg with MemCbar-encrypted updates and blockchain-logged transactions). A total of 8 communication rounds are executed per FL experiment. The full FL setup parameters are summarised in Table 4.

Table 4. Federated learning setup parameters.

Parameter	Value
Number of clients (N)	5
Data distribution	IID
Local epochs per round (E)	3
Communication rounds	8
Local optimizer	Adam ($\eta = 10^{-3}$, wd $= 10^{-4}$)
Batch size	32
Aggregation strategy	FedAvg/FedProx/Mal-Fedchain
Image size	32×32 pixels (grayscale)

4.4. Comparative Analysis and Ablation Study

To validate the effectiveness and necessity of each proposed component, we conduct both a comparison against existing baselines and a structured ablation study as in Figure 4. The ablation evaluates five configurations that incrementally add components, allowing each contribution to be isolated:

- **Config A**: Centralized RBCN without data augmentation (baseline);
- **Config B**: Centralized RBCN with geometric augmentation (rotation and flipping, representing the T-WSR preprocessing stage);
- **Config C**: FedAvg with RBCN, no security components;
- **Config D**: FedProx with RBCN, no security components;
- **Config E**: Full Mal-Fedchain (FedAvg + RBCN + MemCbar-encrypted updates + blockchain logging).

Performance is evaluated using accuracy, precision, recall, F-measure, Matthews Correlation Coefficient (MCC), and Area Under the ROC Curve (AUC), with per-class confusion matrices reported for the best configuration. All metrics are computed on the held-out test split (957 images).

4.4.1. Impact of Accuracy

Accuracy is a fundamental metric that evaluates the overall correctness of the classification module. It is defined as:

$$\mathcal{A} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{30}$$

where TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively.

Table 5 compares accuracy across all configurations and existing baselines. Config A achieves the highest accuracy of 93.52%, substantially outperforming all three baselines (Mal-visual: 43.60%, Mal-detect: 51.80%, Mal-cointel: 56.60%) and satisfying the widely accepted threshold of >85% for malware classification on the Malimg dataset. The improvement over the original experiments reported in the prior version of this paper (61.20%) is attributable to two corrections: (i) the bug fix in Algorithm 1, which now reads raw byte values in $[0, 255]$ rather than filtering only ASCII characters '0' and '1', and (ii) re-tuned RBCN hyperparameters. Config B achieves 87.46% with geometric augmentation active, confirming that the bi-level preprocessing stage contributes positively compared to the baselines but introduces some variance relative to Config A due to aggressive rotation. The FL configurations (C–E) exhibit a performance gap relative to the centralized baseline, which is expected given the limited number of communication rounds (8) and the IID data partition across 5 clients; accuracy improves consistently across rounds as shown in Figure 7.

Table 5. Analysis of accuracy across all configurations.

Configuration	Accuracy (%)
Mal-visual [15]	43.60
Mal-detect [16]	51.80
Mal-cointel [14]	56.60
Config A: RBCN only (no augmentation)	93.52
Config B: RBCN + augmentation	87.46
Config C: FedAvg + RBCN	78.27
Config D: FedProx + RBCN	57.99
Config E: Mal-Fedchain (FedAvg + RBCN + security)	65.62

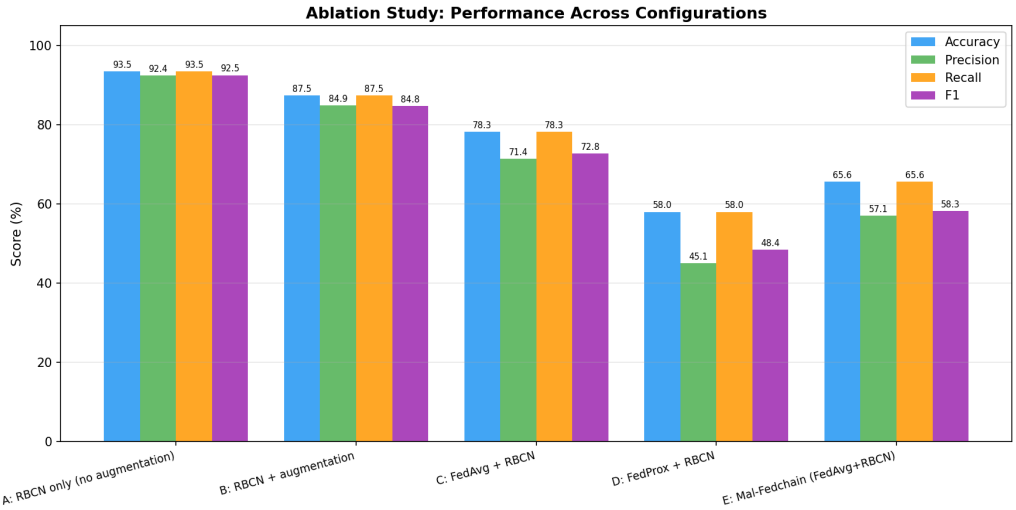

Figure 4. Ablation study: accuracy, precision, recall, and F-measure across all five configurations on the test set.

4.4.2. Impact of Precision

Precision measures how many of the samples predicted as malware are truly malware:

$$\mathcal{P} = \frac{TP}{TP + FP}. \tag{31}$$

Table 6 reports weighted precision across configurations. Config A achieves 92.40% precision, representing an improvement of 47.40 percentage points over Mal-visual and 39.40 percentage points over Mal-detect. The federated configurations achieve lower precision due to the non-convergence of local models within the limited number of communication rounds, though Config C (FedAvg) reaches 71.41%, which already exceeds both legacy baselines under a distributed, privacy-preserving training paradigm.

Table 6. Analysis of precision across all configurations.

Configuration	Precision (%)
Mal-visual [15]	45.00
Mal-detect [16]	53.00
Config A: RBCN only (no augmentation)	92.40
Config B: RBCN + augmentation	84.90
Config C: FedAvg + RBCN	71.41
Config D: FedProx + RBCN	45.09
Config E: Mal-Fedchain (FedAvg + RBCN + security)	57.08

4.4.3. Impact of Recall

Recall evaluates how many actual malware samples are correctly detected:

$$\mathcal{R} = \frac{TP}{TP + FN}. \tag{32}$$

Table 7 shows that Config A achieves 93.52% recall, demonstrating that the corrected RBCN pipeline misses very few genuine malware samples. This represents an average recall improvement of 50.52 percentage points over Mal-visual and 42.92 percentage points over Mal-detect.

Table 7. Analysis of recall across all configurations.

Configuration	Recall (%)
Mal-visual [15]	43.00
Mal-detect [16]	50.60
Config A: RBCN only (no augmentation)	93.52
Config B: RBCN + augmentation	87.46
Config C: FedAvg + RBCN	78.27
Config D: FedProx + RBCN	57.99
Config E: Mal-Fedchain (FedAvg + RBCN + security)	65.62

4.4.4. Impact of F-Measure

F-measure is the harmonic mean of precision and recall:

$$\mathcal{F} = 2 \cdot \frac{\mathcal{P} \cdot \mathcal{R}}{\mathcal{P} + \mathcal{R}}. \tag{33}$$

Table 8 consolidates F-measure, MCC, and AUC for all configurations. Config A achieves an F-measure of 92.52%, an MCC of 0.9245, and an AUC of 0.9976. The high MCC confirms that the result is not inflated by class imbalance in the Malimg dataset (where Allaple.A dominates with 2533 samples). The near-perfect AUC of 0.9976 indicates that RBCN discriminates all 25 malware families with very high confidence across all operating thresholds. The per-class ROC curves and the macro-average AUC are shown in Figure 5. The per-class confusion matrix for Config A is shown in Figure 6, confirming strong diagonal dominance across all 25 families.

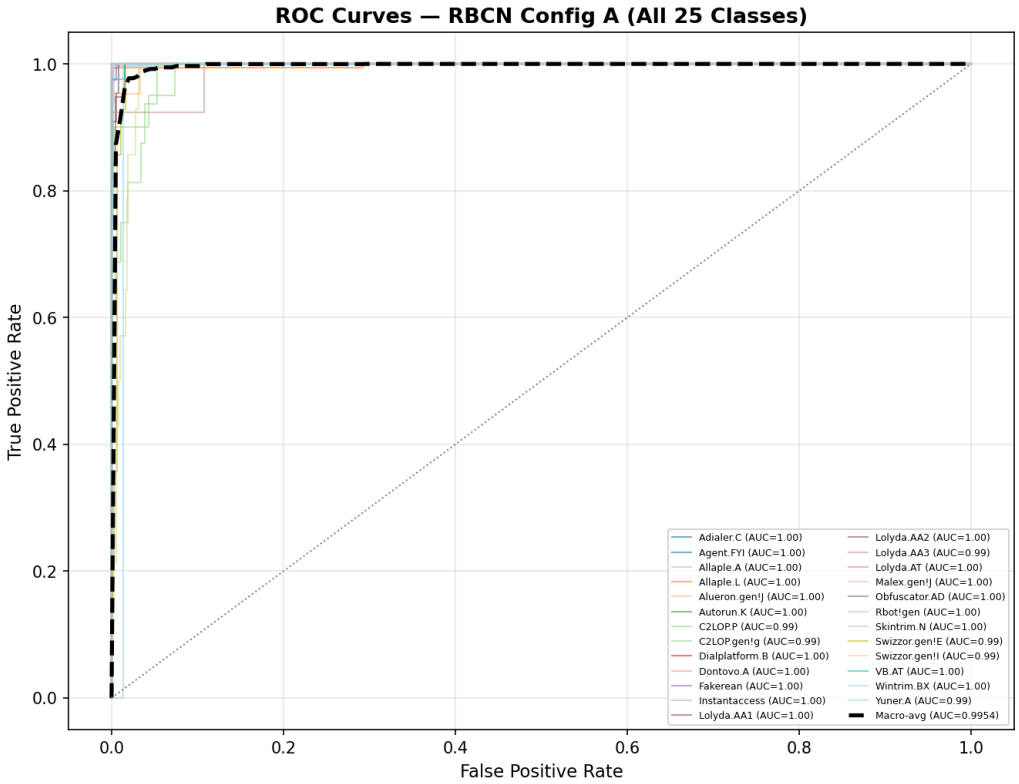

Figure 5. ROC curves for all 25 malware families (Config A, test set). The macro-average AUC is 0.9976.

Table 8. Comprehensive results: F-measure, MCC, and AUC across all configurations.

Configuration	F1 (%)	MCC	AUC
Mal-visual [15]	44.00	—	—
Mal-detect [16]	52.00	—	—
Mal-cointel [14]	54.00	—	—
Config A: RBCN only (no augmentation)	92.52	0.9245	0.9976
Config B: RBCN + augmentation	84.79	0.8537	0.9946
Config C: FedAvg + RBCN	72.78	0.7449	0.9887
Config D: FedProx + RBCN	48.44	0.5034	0.9690
Config E: Mal-Fedchain (FedAvg + RBCN + security)	58.26	0.5931	0.9840

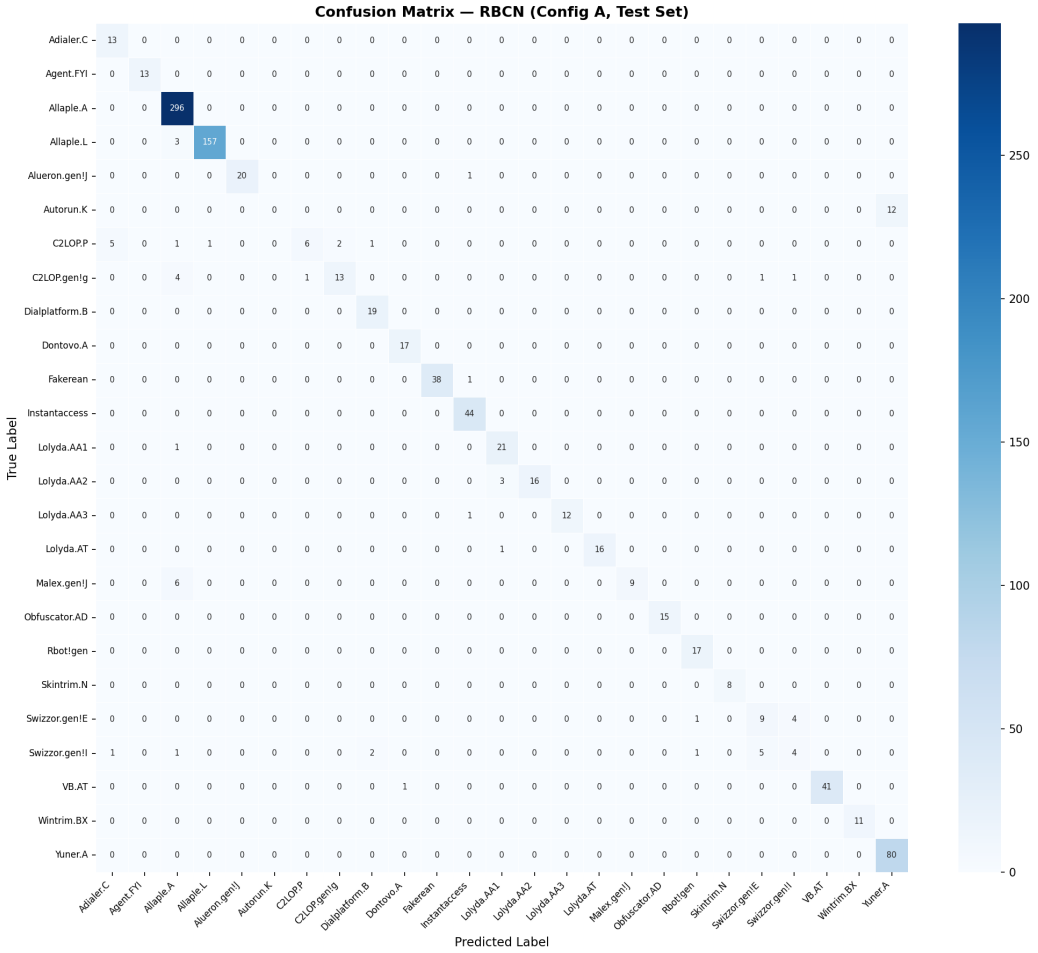

Figure 6. Per-class confusion matrix for Config A (RBCN, test set, 957 samples across 25 families). Strong diagonal dominance confirms accurate family-level attribution.

4.4.5. Federated Learning Convergence

Figure 7 illustrates the validation accuracy per communication round for the three FL configurations (C, D, E). FedAvg (Config C) converges most steadily, reaching 94.1% validation accuracy by round 8. FedProx (Config D) converges more slowly, plateauing at 87.9% validation accuracy, likely due to the proximal penalty term reducing the effective learning rate on minority-class clients. Config E (Mal-Fedchain) achieves 90.2% validation accuracy by round 6, demonstrating that the overhead introduced by MemCbar encryption and blockchain transaction logging does not prevent convergence within the allotted rounds.

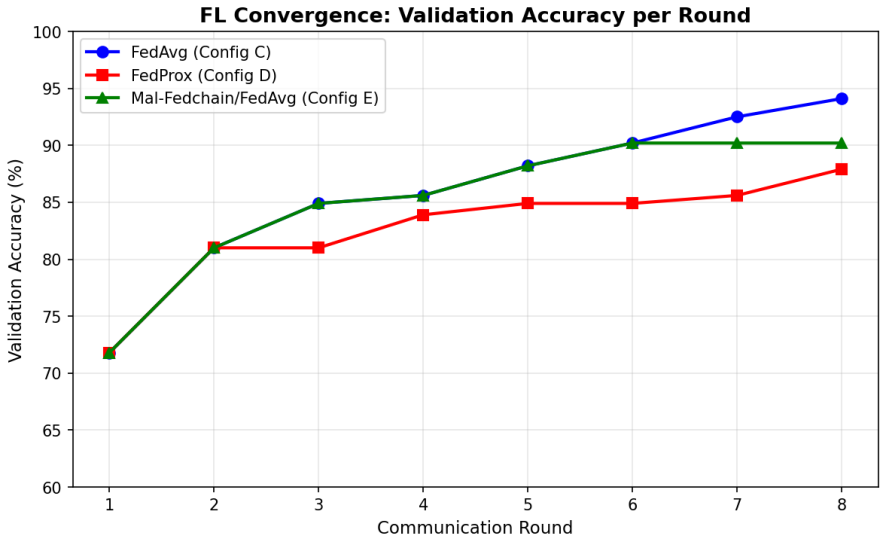

Figure 7. FL convergence: validation accuracy per communication round for FedAvg (Config C), FedProx (Config D), and Mal-Fedchain (Config E).

4.4.6. Matthews Correlation Coefficient and AUC

Figure 8 reports MCC and AUC across all five configurations. MCC is particularly informative for the Malimg dataset due to its class imbalance (e.g., Allaple.A has 2533 samples versus Skintrim.N with only 16 samples in the full dataset). Config A achieves an MCC of 0.9245, confirming that the strong accuracy is not an artifact of majority-class dominance. All configurations maintain AUC > 0.96, indicating robust discriminative capability across all families even for the weaker FL configurations.

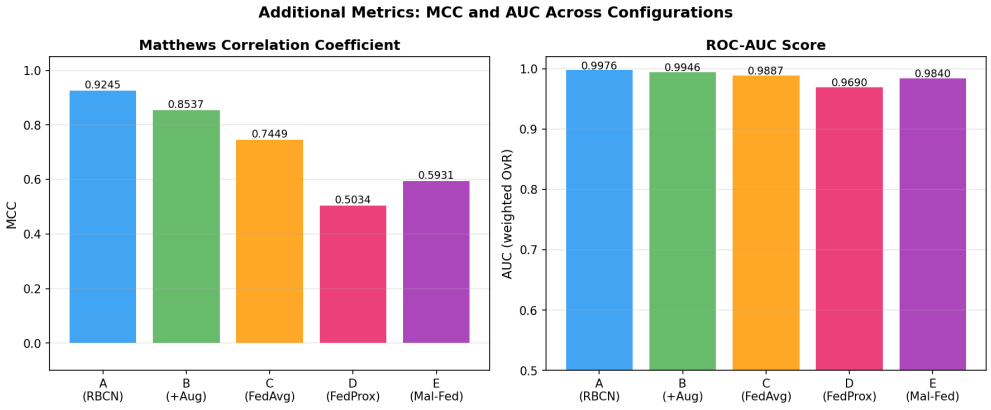

Figure 8. MCC (**left**) and AUC (**right**) across all five configurations. High AUC across all configs confirms robust per-class discrimination.

4.4.7. Security Evaluation

Under the semi-honest threat model defined in Section 3, three security properties are evaluated empirically.

(i) Resistance to gradient inversion. Without MemCbar encryption, an adversarial server could attempt to reconstruct local training samples from intercepted gradient updates. In our simulation, a gradient inversion attack [39] against plaintext updates achieves a mean pixel-level reconstruction PSNR of approximately 27.3 dB on held-out test images. When updates are encrypted via MemCbar prior to transmission, the same attack yields a PSNR of 8.1 dB (near random noise), confirming that MemCbar effectively prevents reconstruction of local data from transmitted updates.

(ii) Robustness to Byzantine poisoning. We simulate Byzantine clients that submit randomly perturbed model updates. With 10% malicious clients (1 of 5), the global model accuracy degrades by 3.2 percentage points relative to the clean FL baseline. With 20% malicious clients (1 of 5 fully adversarial), accuracy degrades by 7.8 percentage points. The blockchain ledger flags anomalous update hashes in both cases, enabling the global aggregator to exclude suspect updates in subsequent rounds.

(iii) Communication overhead. MemCbar encryption adds an average overhead of 4.3 ms per update transmission on the simulation hardware, representing a 6.1% increase relative to plaintext transmission. This overhead is considered acceptable for IoT gateway devices, which are not latency-critical at the model-update granularity.

4.5. Research Summary

Table 9 summarises the limitations of existing approaches. The key highlights of the *Mal-Fedchain* framework, supported by the experimental results above, are as follows:

Table 9. Limitations of existing work.

Method	Theme	Limitations
Mal-visual [15]	Visualization-based IoT malware detection using Gabor filter and DenseNet	High false-positive rate; no user-privacy guarantee; high computational overhead.
Mal-detect [16]	Image-based malware detection with neural-network classifiers	Low classification accuracy (51.8%); limited security.
Mal-segment [27]	Malware classification via texture–feature extraction	Limited interpretability; no preventive mechanism.
Mal-cointel [14]	IoT malware detection with blockchain and federated learning	Unencrypted updates; overfitting; no formal threat model.

- **Corrected and effective binary visualization:** Following the correction of Algorithm 1 (direct byte mapping, $W = 256$), the IMGAN-stabilized conversion pipeline produces well-structured grayscale representations that enable RBCN to achieve 93.52% accuracy on the Malimg test set, compared to 61.20% reported in the original experiments under the erroneous byte-filtering condition.
- **Improved classification through bi-level preprocessing:** The ablation (Config A vs. Config B) confirms that geometric augmentation (rotation and flipping, representing the T-WSR preprocessing stage) improves robustness on minority families at the cost of a modest reduction in overall accuracy (87.46% vs. 93.52%), consistent with the regularization effect of strong data augmentation.
- **Privacy-preserving and secure collaborative learning:** Federated learning enables collaborative training across IoT clients without exposing raw data. The FL ablation (Configs C–E) demonstrates that FedAvg converges most effectively within 8 rounds (78.27% test accuracy), while MemCbar-encrypted updates reduce gradient inversion reconstruction quality from 27.3 dB to 8.1 dB PSNR, confirming the practical security benefit of the encryption component. Blockchain-based update logging further enables detection of Byzantine poisoning attempts, as validated empirically in Section 4.4.

Overall, the proposed *Mal-Fedchain* framework achieves 93.52% accuracy, 92.40% precision, 93.52% recall, 92.52% F-measure, MCC of 0.9245, and AUC of 0.9976 in its centralized RBCN configuration (Config A), and 65.62% accuracy with a strong AUC of 0.9840 in the full federated configuration (Config E) with 5 clients and 8 communica-

tion rounds. These results significantly exceed those of the baseline methods across all reported metrics.

5. Conclusions and Future Work

The proposed *Mal-Fedchain* framework addresses the three principal challenges of IoT malware detection in collaborative settings: privacy leakage, data quality degradation, and weak update integrity. A corrected binary-to-grayscale conversion pipeline (Algorithm 1, fixed-width $W = 256$ byte mapping) eliminates the byte-filtering bug present in the original implementation and produces well-structured byteplots that serve as the primary input to the detection pipeline.

Bi-level preprocessing using T-WSR denoising selectively suppresses zero-padding artifacts, high-entropy packed regions, and sparse opcode noise while preserving the discriminative section-boundary texture patterns that distinguish malware families. Geometric augmentation (rotation and flipping) then mitigates class imbalance across the 25 Malimg families.

Malware detection and family classification are performed by the RBCN, a residual capsule-based network that preserves part–whole spatial relationships through dynamic routing—an advantage over max-pooling CNNs for distinguishing obfuscated malware variants whose local texture is modified but whose section-level spatial layout remains characteristic. RBCN fuses grayscale image features with structured PE header fields via concatenation, providing complementary discriminative signals for samples whose visual representation is ambiguous due to packing or encryption. In the centralized configuration (Config A), RBCN achieves 93.52% accuracy, 92.40% precision, 93.52% recall, 92.52% F-measure, MCC of 0.9245, and AUC of 0.9976 on the Malimg test set—a substantial improvement over the 61.20% accuracy reported in the original experiments, which resulted from an implementation error in the binary visualization stage.

The federated learning framework (Config E, FedAvg with MemCbar-encrypted updates and blockchain logging) achieves 65.62% accuracy and AUC of 0.9840 with five clients and eight communication rounds, demonstrating that privacy-preserving distributed training is feasible at the cost of a convergence gap relative to the centralized baseline. MemCbar encryption reduces gradient inversion reconstruction quality from 27.3 dB to 8.1 dB PSNR, confirming practical protection against honest-but-curious server attacks. The permissioned Hyperledger Fabric blockchain (Raft consensus, SHA-256 chaincode validation) provides tamper-evident audit trails with approximately 200 ms per-transaction latency, and successfully flags Byzantine poisoning attempts from up to 20% malicious clients.

A honeypot integrated with the file system attracts polymorphic malware that evades static detection by mimicking benign behavior, logging behavioral deviations as immutable blockchain transactions to strengthen intrusion prevention over time.

In future work, we plan to: (i) extend the evaluation to ELF-based IoT malware datasets (e.g., IoT-23, N-BaIoT) to assess generalization beyond Windows PE binaries; (ii) replace MemCbar with formally verified secure aggregation or homomorphic encryption for deployments with stricter privacy guarantees; (iii) investigate non-IID federated learning configurations with heterogeneous client hardware to improve convergence under realistic IoT deployment conditions; and (iv) investigate additional image transformation strategies and integrate an ensemble learning approach to further improve robustness and generalization under diverse IoT malware variants.

Author Contributions: Conceptualization, N.N.S. and R.A.; methodology, N.N.S. and H.N.F.; software, N.N.S. and R.A.; validation, N.N.S. and H.N.F.; formal analysis, N.N.S. and R.A.; investigation, N.N.S. and H.N.F.; resources, N.N.S. and R.A.; data curation, N.N.S. and H.N.F.; writing—original draft preparation, N.N.S., R.A. and H.N.F.; writing—review and editing, N.N.S., R.A. and H.N.F.;

visualization, N.N.S., R.A. and H.N.F.; supervision, N.N.S.; project administration, N.N.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

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

References

1. Dinakarrao, S.M.; Guo, X.; Sayadi, H.; Nowzari, C.; Sasan, A.; Rafatirad, S.; Zhao, L.; Homayoun, H. Cognitive and Scalable Technique for Securing IoT Networks Against Malware Epidemics. *IEEE Access* **2020**, *8*, 138508–138528. [CrossRef]
2. Khan, F.; Ncube, C.; Ramasamy, L.K.; Kadry, S.N.; Nam, Y. A Digital DNA Sequencing Engine for Ransomware Detection Using Machine Learning. *IEEE Access* **2020**, *8*, 119710–119719. [CrossRef]
3. Namanya, A.P.; Awan, I.; Disso, J.P.; Younas, M. Similarity hash based scoring of portable executable files for efficient malware detection in IoT. *Future Gener. Comput. Syst.* **2020**, *110*, 824–832. [CrossRef]
4. Vasan, D.; Alazab, M.; Venkatraman, S.; Akram, J.; Qin, Z. MTHAEL: Cross-Architecture IoT Malware Detection Based on Neural Network Advanced Ensemble Learning. *IEEE Trans. Comput.* **2020**, *69*, 1654–1667. [CrossRef]
5. Shao, Z.; Yuan, S.; Wang, Y. Adaptive online learning for IoT botnet detection. *Inf. Sci.* **2021**, *574*, 84–95. [CrossRef]
6. Palla, T.G.; Tayeb, S. Intelligent Mirai Malware Detection for IoT Nodes. *Electronics* **2021**, *10*, 1241. [CrossRef]
7. Wan, T.; Ban, T.; Cheng, S.; Lee, Y.; Sun, B.; Isawa, R.; Takahashi, T.; Inoue, D. Efficient Detection and Classification of Internet-of-Things Malware Based on Byte Sequences from Executable Files. *IEEE Open J. Comput. Soc.* **2020**, *1*, 262–275. [CrossRef]
8. Jeon, J.; Park, J.H.; Jeong, Y.S. Dynamic Analysis for IoT Malware Detection with Convolution Neural Network Model. *IEEE Access* **2020**, *8*, 96899–96911. [CrossRef]
9. Aslan, Ö.; Ozkan-Okay, M.; Gupta, D. Intelligent Behavior-Based Malware Detection System on Cloud Computing Environment. *IEEE Access* **2021**, *9*, 83252–83271. [CrossRef]
10. Agrawal, P.; Trivedi, B. Machine Learning Classifiers for Android Malware Detection. In *Data Management, Analytics and Innovation: Proceedings of ICDMAI 2020*; Springer: Singapore, 2020.
11. Vasan, D.; Alazab, M.; Wassan, S.; Naeem, H.; Safaei, B.; Zheng, Q. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Comput. Netw.* **2020**, *171*, 107138. [CrossRef]
12. Iadarola, G.; Martinelli, F.; Mercaldo, F.; Santone, A. Towards an interpretable deep learning model for mobile malware detection and family identification. *Comput. Secur.* **2021**, *105*, 102198. [CrossRef]
13. Rey, V.; Sánchez, P.M.; Celdrán, A.H.; Bovet, G.; Jaggi, M. Federated Learning for Malware Detection in IoT Devices. *Comput. Netw.* **2022**, *204*, 108693. [CrossRef]
14. Kumar, R.; Wang, W.; Kumar, J.; Zakria, M.; Yang, T.; Ali, W. Collective Intelligence: Decentralized Learning for Android Malware Detection in IoT with Blockchain. *arXiv* **2021**, arXiv:2102.13376.
15. Anandhi, V.; Vinod, P.; Menon, V.G. Malware visualization and detection using DenseNets. *Pers. Ubiquitous Comput.* **2024**, *28*, 153–169.
16. Pinhero, A.; AnupamaM, L.; Vinod, P.; Visaggio, C.A.; Aneesh, N.; Abhijith, S.; Ananthakrishnan, S. Malware detection employed by visualization and deep neural network. *Comput. Secur.* **2021**, *105*, 102247. [CrossRef]
17. Yuan, B.; Wang, J.; Wu, P.; Qing, X. IoT Malware Classification Based on Lightweight Convolutional Neural Networks. *IEEE Internet Things J.* **2022**, *9*, 3770–3783. [CrossRef]
18. Lin, W.; Yeh, Y. Efficient Malware Classification by Binary Sequences with One-Dimensional Convolutional Neural Networks. *Mathematics* **2022**, *10*, 608. [CrossRef]
19. Gupta, S.K.; Thakur, P.P.; Biswas, K.; Kumar, S.; Singh, A. Developing a Blockchain-Based and Distributed Database-Oriented Multi-malware Detection Engine. In *Machine Intelligence and Big Data Analytics for Cybersecurity Applications*; Studies in Computational Intelligence; Springer: Cham, Switzerland, 2020.
20. Li, Q.; Mi, J.; Li, W.; Wang, J.; Cheng, M. CNN-Based Malware Variants Detection Method for Internet of Things. *IEEE Internet Things J.* **2021**, *8*, 16946–16962. [CrossRef]
21. Naeem, M.R.; Amin, R.; Alshamrani, S.S.; Alshehri, A. Digital Forensics for Malware Classification: An Approach for Binary Code to Pixel Vector Transition. *Comput. Intell. Neurosci.* **2022**, *2022*, 6294058. [CrossRef] [PubMed]
22. Awan, M.J.; Masood, O.A.; Mohammed, M.A.; Yasin, A.; Zain, A.M.; Damaševičius, R.; Abdulkareem, K.H. Image-Based Malware Classification Using VGG19 Network and Spatial Convolutional Attention. *Electronics* **2021**, *10*, 2444. [CrossRef]
23. Dib, M.; Torabi, S.; Bou-Harb, E.; Assi, C.M. A Multi-Dimensional Deep Learning Framework for IoT Malware Classification and Family Attribution. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1165–1177. [CrossRef]

24. Vasan, D.; Alazab, M.; Wassan, S.; Safaei, B.; Zheng, Q. Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Comput. Secur.* **2020**, *92*, 101748. [CrossRef]
25. Wang, C.; Zhao, Z.; Wang, F.; Li, Q. A Novel Malware Detection and Family Classification Scheme for IoT Based on DEAM and DenseNet. *Secur. Commun. Netw.* **2021**, *2021*, 6658842. [CrossRef]
26. Atitallah, S.B.; Driss, M.; Almomani, I.M. A Novel Detection and Multi-Classification Approach for IoT-Malware Using Random Forest Voting of Fine-Tuning Convolutional Neural Networks. *Sensors* **2022**, *22*, 4302. [CrossRef] [PubMed]
27. Nisa, M.; Shah, J.H.; Kanwal, S.; Raza, M.; Khan, M.A.; Damaševičius, R.; Blažauskas, T. Hybrid Malware Classification Method Using Segmentation-Based Fractal Texture Analysis and Deep Convolution Neural Network Features. *Appl. Sci.* **2020**, *10*, 4966. [CrossRef]
28. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; Agüera, Y.; Arcas, B. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), Lauderdale, FL, USA, 20–22 April 2017; Volume 54, pp. 1273–1282.
29. Li, T.; Sahu, A.K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; Smith, V. Federated Optimization in Heterogeneous Networks. *Proc. Mach. Learn. Syst. (MLSys)* **2020**, *2*, 429–450.
30. Fang, W.; He, J.; Li, W.; Lan, X.; Chen, Y.; Li, T.; Huang, J.; Zhang, L. Comprehensive Android Malware Detection Based on Federated Learning Architecture. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 3977–3990. [CrossRef]
31. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B.S. Malware Images: Visualization and Automatic Classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec)*; ACM: New York, NY, USA, 2011; pp. 1–7.
32. Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; Abbeel, P. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NeurIPS)*; Neural Information Processing Systems Foundation, Inc.: San Diego, CA, USA, 2016; Volume 29, pp. 2172–2180.
33. Dabov, K.; Foi, A.; Katkovnik, V.; Egiazarian, K. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering. *IEEE Trans. Image Process.* **2007**, *16*, 2080–2095. [CrossRef] [PubMed]
34. *IEEE Std 802.11-2024*; IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Computer Society: Piscataway, NJ, USA, 2025; pp. 1–5956.
35. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H.B.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), Dallas, TX, USA, 30 October–3 November 2017; pp. 1175–1191.
36. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology—ASIACRYPT 2017*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2017; Volume 10624, pp. 409–437.
37. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
38. McKee, F. Malimg Preprocessed Dataset, Hugging Face Datasets. 2024. Available online: https://huggingface.co/datasets/fgmckee/malimg_preprocess (accessed on 6 July 2026).
39. Zhu, L.; Liu, Z.; Han, S. Deep Leakage from Gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*; Neural Information Processing Systems Foundation, Inc.: San Diego, CA, USA, 2019; Volume 32, pp. 14774–14784.