Annals of Emerging Technologies in Computing (AETiC)
Vol. 10, No. 3, 2026

Research Article

Intelligent Scanning and Remediation Algorithms for Information Security Vulnerabilities in IoT Terminals

Pengcheng He* and Weiju Kong

Xinyang Vocational and Technical College, Xinyang, China
17630988594@163.com; kong_weiju@163.com
*Correspondence: 17630988594@163.com

Received: 19 January 2025; Accepted: 27 June 2026; Published: 1 July 2026

Abstract: The number of IoT terminal devices is growing rapidly. However, due to limited computing resources, diverse architectures, and difficulties in updating algorithm models, traditional vulnerability detection and fault repair mechanisms are inapplicable. Therefore, this paper constructs a security vulnerability scanning algorithm for IoT terminals. The IV-PARM model constructed in this paper combines multiple features, including data from three modalities: firmware semantics, runtime behavior, and network traffic, to build a heterogeneous graph neural network. Device vulnerability association inference, combined with risk perception, is generated based on a reinforcement learning model. An adaptive and lightweight repair strategy is used to guide the update of IoT terminals. The model framework of this paper adopts a three-level collaborative approach, which can reduce the resource overhead of the terminal for vulnerability detection while ensuring the accuracy of vulnerability detection. This paper conducts experiments on 1200 real samples and 120,000 behavior logs. The experimental results show that the IV-PARM model can achieve an accuracy of 85.7% in vulnerability detection, with a false positive rate of only 2.3% and a repair success rate as high as 91%. On the other hand, in terms of latency, the average latency of the model is less than 45 milliseconds, and the memory usage is less than 15 MB. The experimental results in this paper demonstrate that the proposed model has significant advantages in terms of effectiveness, robustness, and inference.

Keywords: ***Adaptive repair; Graph neural network; IoT security; Intelligent vulnerability detection; Reinforcement learning***

1. Introduction

The Internet of Things (IoT) has been widely used in society and integrated into multiple fields such as industrial manufacturing, medical health, and home. By 2025, the number of IoT devices will increase to 30 billion. Such a huge number will form a highly interactive, interconnected, and data-driven ecosystem. While enjoying the efficient and intelligent services brought by the IoT, the security threats hidden behind the IoT are becoming increasingly apparent. IoT devices have a common problem of limited resources. The computing power of IoT devices is weak, the storage space is limited, and the energy supply is insufficient [1]. At the same time, a large number of IoT terminals are distributed in complex and diverse environments, with long life cycles [2]. Hardware updates are also relatively difficult. Therefore, IoT terminals have become the primary target of network attackers. As a result, a large number of attacks in recent years, such as botnets and Bluetooth vulnerabilities, are large-scale security problems caused by IoT devices. These threats will not only cause the leakage of user privacy and loss of property, but also threaten national and social security. Traditional vulnerability detection methods, such as intrusion detection systems and static code analysis tools, although they have reached a mature level in the computing environment, face challenges in the new network architecture of the IoT [3]. Some problematic IoT devices lack a unified

Pengcheng He and Weiju Kong, "Intelligent Scanning and Remediation Algorithms for Information Security Vulnerabilities in IoT Terminals", *Annals of Emerging Technologies in Computing (AETiC)*, Print ISSN: 2516-0281, Online ISSN: 2516-029X, 1 July 2026, Vol. 10, No. 3, pp. 50-66, Published by International Association for Educators and Researchers (IAER), DOI: 10.33166/AETiC.2026.03.004, Available: http://aetic.theiaer.org/archive/v10/v10n3/p4.html.

interface or remote management system. This creates a bottleneck in the proactive detection and analysis of vulnerabilities by intelligent algorithms.

Therefore, in this context, we need to develop a security vulnerability repair mechanism for IoT terminals that is intelligent. This mechanism should have the ability to autonomously perceive and identify security vulnerabilities in an intelligent way, and accurately locate and respond to them [4]. In recent years, a large number of artificial intelligence methods have emerged, and intelligent vulnerability scanning models can be built by combining AI algorithms. These models can automatically obtain certain patterns from firmware, images, network traffic and other data, and thus analyze and predict unknown vulnerabilities [5]. For example, natural language processing technology can be used to identify and parse strings, configuration files and other data in components at the semantic level, thereby discovering hard-coded credentials. Graph neural networks can also be used to model the control flow level of the binary code of firmware, so that backdoors or malicious activities in network communication can be discovered without prior knowledge. However, if secure closed-loop repair cannot be achieved solely through intelligent scanning, for resource-constrained IoT terminals, we not only need to build intelligent scanning, but also a lightweight, adaptive intelligent repair strategy. This intelligent repair strategy often differs from traditional patching methods. It requires implementing runtime protection strategies using virtual patches, capable of intercepting malicious input without modifying the firmware. Simultaneously, it employs containerization or microkernel architectures to achieve modular isolation, thus preventing the vulnerability from being contained within its impact range [6, 7]. Our framework must balance the practicality and security of the model to avoid service interruptions due to over-protection or device malfunctions.

2. Literature Review

2.1. Limitations of Traditional IoT Vulnerability Detection

Traditional IoT vulnerability detection methods, such as static analysis, fuzzing, and dynamic taint analysis, are mature in general computing environments but face fundamental challenges when directly applied to IoT terminals. The core contradiction lies between the complexity of detection methods and the extreme resource constraints of the terminals. For example, automated firmware emulation (e.g., FirmAE [8]) and symbolic execution (e.g., the framework by Touqir *et al*. [9]) can perform in-depth firmware analysis, but they incur substantial computational overhead, suffer from low boot success rates, and heavily depend on specific runtime environments (e.g., a full Linux kernel), making them difficult to deploy in real-time on resource-constrained or non-standard architecture terminals [10]. Similarly, state machine-based fuzzing targeting protocols [11] can discover communication interface vulnerabilities, but its test case generation and execution typically rely on external, powerful computing platforms and cannot be integrated into the terminals themselves. These methods are essentially "offline" or "bypass" analyses, failing to address the need for lightweight, real-time detection at the device's operational site. Their fundamental limitation is the lack of generalization ability: detection logic tailored for specific firmware or protocols struggles to adapt to the vast fragmentation in IoT device models, architectures, and software versions.

2.2. Advancements and Gaps in AI-Based Vulnerability Identification

To address these challenges, academia has turned to artificial intelligence-based methods, aiming to automatically learn vulnerability patterns from data. This research can be broadly categorized into three types, each with significant limitations:

Code Representation-Based Vulnerability Detection: This line of work (e.g., [12-14]) attempts to learn vulnerability features from binary code or control flow graphs (CFG). For instance, treating instructions as images [12] or using Graph Neural Networks (GNNs) to analyze function-level CFGs [14]. While progress has been made on specific datasets, their effectiveness critically depends on high-quality, cross-architecture code representation learning. A key underexplored flaw is that most methods are validated only on single vulnerability types (e.g., buffer overflows) or homogeneous architecture datasets; their model's generalization performance sharply declines when confronted with the diverse vulnerability types (from

memory errors to logic flaws) and heterogeneous hardware (ARM, MIPS, RISC-V) in real IoT environments [10]. This reveals the inherent insufficiency of current AI models in cross-task, cross-domain generalization.

Behavior and Traffic-Based Anomaly Detection: Another category of work (e.g., [15, 16]) detects anomalies by monitoring device runtime behavior (system calls) or network traffic using sequential models (e.g., LSTM). This approach avoids complex code analysis and has greater deployment potential. However, they typically can only raise an "anomaly detected" alarm without precisely locating the specific vulnerable code location or type—knowing the "what" but not the "why"—which complicates subsequent remediation actions. Furthermore, the computational and energy overhead of continuously running complex deep learning models (e.g., LSTM) for real-time traffic analysis on resource-constrained terminals is itself an unresolved contradiction.

Privacy-Preserving Collaborative Learning Frameworks: To leverage distributed terminal data while protecting privacy, frameworks like federated learning have been introduced [17]. While a promising direction, existing research mostly focuses on model training efficacy under privacy constraints, often overlooking the real-time, lightweight requirements for model inference on edge devices. A federated learning model that performs well on a server may have a parameter count and computational complexity that renders it completely infeasible to run on terminal devices.

2.3. The Missing Link: From Detection to Adaptive, Closed-Loop Remediation

It is noteworthy that the vast majority of the aforementioned research stops at "detection." Although a few works have begun exploring self-healing for IoT devices (e.g., container-based isolation, virtual patching), these attempts are often passive and rule-based, lacking an intelligent closed-loop capable of adaptive decision-making based on real-time risk assessment, device context, and resource status. For instance, a simple rule might command all devices to disable a certain service, but this indiscriminately affects devices with high availability requirements, causing service interruptions. This represents a core gap in current research: how to integrate high-precision vulnerability perception, cross-device relational reasoning, and lightweight, risk-adaptive remediation decision-making into a unified system that operates autonomously within resource boundaries.

2.4. Emerging Paradigms and the Position of Our Work

Recently, two emerging paradigms are noteworthy, yet neither fully addresses the aforementioned closed-loop problem:

Automated Program Repair (APR): In software engineering, APR techniques aim to automatically generate patches. However, traditional APR techniques heavily rely on source code and test suites, both of which are typically unavailable in the proprietary, binary firmware environment of IoT, hindering their direct application.

Large Language Models (LLMs) for Vulnerability Analysis: LLMs like CodeBERT and the GPT series demonstrate powerful code understanding capabilities and are beginning to be used for vulnerability detection. However, deploying these models with billions of parameters onto IoT terminals is absolutely infeasible. While cloud-based LLMs could be accessed via APIs, this introduces latency, network dependency, and privacy leakage risks.

Therefore, the position of our work (IV-PARM) is not to simply apply the latest, most complex models, but to perform a critical systemic integration and innovative trade-off. We acknowledge and confront the fundamental contradiction between the computational demands of advanced AI models and the resource constraints of IoT terminals. Our contribution lies in: 1) Offloading compute-intensive tasks (feature extraction, GNN inference) via an edge collaborative architecture, leaving terminals responsible only for lightweight data collection; 2) Designing a lightweight heterogeneous graph neural network, optimized for resource-constrained environments and integrating multimodal features, for accurate and generalizable vulnerability correlation detection, rather than directly deploying giant LLMs; 3) Introducing a reinforcement learning-based decision module to generate risk-adaptive remediation strategies at the edge, thereby constructing for the first time a complete intelligent security closed-loop from perception, analysis, decision to execution, bridging the gap between current vulnerability detection technologies and executable, acceptable automated remediation.

3. Information Security Vulnerabilities in IoT Terminals

3.1. Overall Model Architecture

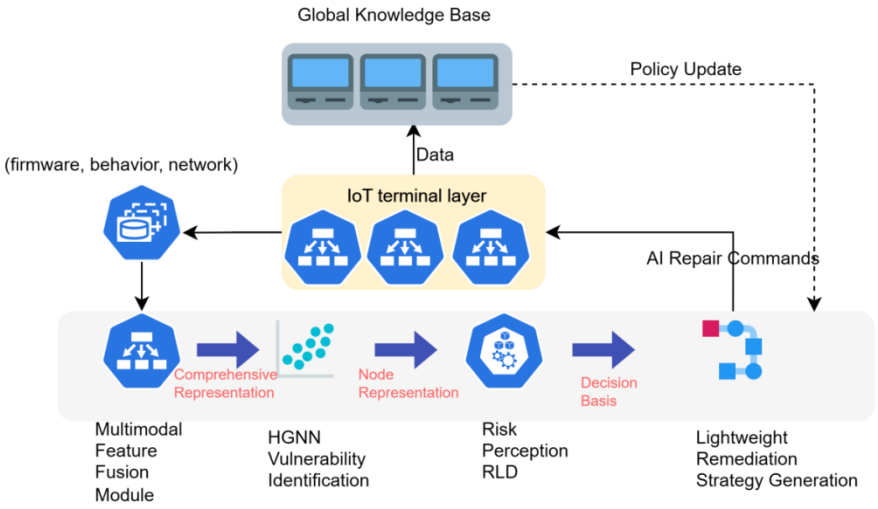

Figure 1. Model framework

As shown in Figure 1, we have constructed a vulnerability awareness and remediation model for IoT terminals. Unlike traditional vulnerability scanning tools that rely on manual testing or signatures, we have constructed a high-coverage, low-overhead method. The method in this paper achieves an intelligent closed-loop from data to feedback. The model constructed in this paper is called IV-PARM. By continuously sensing, identifying and executing processes, feedback is obtained during the process, thereby further updating the model. This process framework integrates multiple technical systems such as artificial intelligence program analysis. In summary, we have adopted a multi-layer architecture. Our terminal deployment layer uses lightweight agents with storage space of less than 50 KB. These agents are only responsible for collecting raw data, while the edge layer is responsible for core data computing tasks, including feature extraction graphs, data modeling and policy reasoning. The cloud storage center is responsible for global knowledge reasoning and stores the global knowledge base, which is used for continuous model learning and policy optimization. This design avoids the resource shortage caused by directly deploying complex algorithms on the terminal. It also avoids the data bandwidth and privacy overhead caused by transmitting large amounts of data to the cloud [18, 19]. Specifically, we assume that the IoT environment contains n active terminal devices, which constitute a device set $\mathrm{D} = d_1, d_2, \ldots, d_N$. For any one of these devices $d_i \in \mathrm{D}$, we describe its security characteristics using three types of data. We describe its security characteristics as a triple, as shown in Equation (1)

$$\mathbf{s}_i = (\mathbf{f}_i, \mathbf{b}_i, \mathbf{n}_i) \tag{1}$$

In $\mathbf{f}_i \in R^{L_f}$, Equation (1) is used to represent semantic-level code feature vectors extracted from the firmware of a device. The dimension of the code feature vector is usually set to 512 dimensions. These feature vectors are used to reflect potential logical design defects in the device, such as command injection or buffer overflow. $\mathbf{b}_i \in R^{L_b}$ represents a behavioral feature vector generated by the device during operation. The dimension is generally 64 dimensions, which includes system calls, the frequency of sequences, the rate of process creation, the CPU utilization, and file access patterns. $\mathbf{n}_i \in R^{L_n}$ represents some traffic-related feature vectors of the device in network communication. The dimension is 32 dimensions. The core of the learning model is to learn a joint mapping function [20], as shown in Equation (2).

$$\mathrm{M} : \mathbf{s}_i \mapsto (\hat{y}_i, a_i^*) \tag{2}$$

In Equation (2), we use $(\hat{y}_i \in 0,1)$ as an output representation of a binary classification to indicate whether a device has a high-risk vulnerability that requires immediate attention. 1 indicates the vulnerability exists, and 0 indicates that there is currently no significant risk. $a_i^* \in \mathrm{A}$ also represents selecting the optimal remediation action from a potentially predefined action space. An example of the action space is shown in Equation (3).

$$\begin{aligned}&(\mathrm{A} = a_1 = \text{Injection input filtering rule},\\ &a_2 = \text{Disable Telnet/SSH service},\\ &a_3 = \text{Trigger differential OTA firmware update},\\ &a_4 = \text{Temporarily isolate to secure VLAN})\end{aligned} \tag{3}$$

The model framework in this paper consists of four modules: a multimodal feature fusion module for obtaining a comprehensive feature vector; a heterogeneous graph neural network vulnerability identification module for core vulnerability identification, risk perception, and reinforcement learning decision-making, used to propose further remediation schemes; and a lightweight remediation execution module, which serves as the action execution module for the reinforcement learning module.

The core innovation of IV-PARM lies in constructing an end-to-end intelligent security closed-loop system. It is the first to organically integrate three key dimensions: multimodal perception, heterogeneous graph-based relational inference, and risk-adaptive decision-making. The multimodal features provide comprehensive security posture awareness, the heterogeneous graph neural network enables cross-device knowledge transfer for detecting unknown vulnerabilities, and the reinforcement learning module generates optimal remediation strategies based on real-time risk assessment. These three components are unified and coordinated through an edge-collaborative architecture, forming a complete autonomous system encompassing perception, analysis, decision-making, and execution.

3.2. Multimodal Feature Fusion Module

A single data source cannot fully reflect the security functions of an IoT terminal. Therefore, we constructed an embedding model that comprehensively analyzes the terminal firmware, behavior network, and other dimensions. First, regarding the embedding of firmware semantics, we utilized an open-source disassembler framework. This framework was used to perform static analysis on the model's firmware, extracting all identifiable functions and forming a function set, as shown in Equation (4).

$$\mathrm{F}\,i = fi1, f_{i2}, \ldots, f_{iK}) \tag{4}$$

In Equation (4), we use K to represent the total number of functions, and each function is represented by one assembly instruction, as shown in Formula (5).

$$\mathbf{x}ij = [xij1, x_{ij2}, \ldots, x_{ijT}] \tag{5}$$

In Equation (5), x_{ijt}, the instruction t-th of a function memory is used to represent the instruction, where T represents the specified maximum length.

We use a bidirectional long short-term memory neural network to encode the contextual dependencies between instructions, specifically using Equation (6).

$$\begin{aligned}&\vec{\mathbf{h}}ijt = \mathrm{LSTMforward}\left(\mathbf{e}(x_{ijt}), \vec{\mathbf{h}}ij, t-1\right)\\ &= \mathrm{LSTMbackward}\left(\mathbf{e}(xijt), \overleftarrow{\mathbf{h}}ij, t+1\right)\\ &= \left[\vec{\mathbf{h}}ijt; \overleftarrow{\mathbf{h}}ijt\right]\end{aligned} \tag{6}$$

In Equation (6), $\mathbf{e}(x_{ijt}) \in R^{d_e}$ represents x_{ijt}, a vector representation of a given instruction obtained after passing through the embedding layer. We set the embedding dimension of the vector to 64. $\vec{\mathbf{h}}ijt \in R^{d_h}$ and $\overleftarrow{\mathbf{h}}ijt \in R^{d_h}$ are used to represent the forward and backward hidden states of a long short-term neural network at a certain time t during the embedding process. d_h represents the total number of hidden units. $[;]$ is used to represent the feature vector concatenation method. $\mathbf{h}_{ijt} \in R^{2d_h}$ represents the final fused feature vector. Since the feature vectors constructed from statistical features of behavior and the network vary greatly, we standardize them uniformly, as shown in Equation (7).

$$\mathbf{b}_i = \frac{\mathbf{b}_i - \boldsymbol{\mu}_b}{\boldsymbol{\sigma}_b}, \quad \mathbf{n}_i = \frac{\mathbf{n}_i - \boldsymbol{\mu}_n}{\boldsymbol{\sigma}_n} \tag{7}$$

In Equation (7), $(\boldsymbol{\mu}_b \in R^{L_b})$ $(\boldsymbol{\mu}_n \in R^{L_n})$ represent device-related behavioral features and network traffic-related features in the training set, respectively. We need to perform a unified mapping on the first three types of features. We use a two-layer perceptron for non-linear transformation, as shown in Equation (8).

$$\mathbf{z}_i = \left[\mathrm{MLP}_f(\mathbf{g}_i); \mathrm{MLP}_b(\mathbf{b}_i); \mathrm{MLP}_n(\mathbf{n}_i) \right] \tag{8}$$

In Equation (8), $\mathrm{MLP}_f : R^{2d_h} \to R^{d_z}$, $\mathrm{MLP}_b : R^{L_b} \to R^{d_z}$ represent the hidden and output layer vectors, using a 256-dimensional representation. $\mathbf{z}_i \in R^{3d_z}$ represents $\mathrm{MLP}_n : R^{L_n} \to R^{d_z}$, the comprehensive feature vector of a given device, which can be used as an input to a subsequent neural network.

Specifically, we used Ghidra as our core disassembly and analysis engine. Ghidra is a powerful, open-source, and scalable software reverse engineering framework released by the National Security Agency (NSA). We utilized its Headless Analyzer interface to automatically unpack firmware images, identify functions, and restore control flow. For each extracted function, we obtained its assembly instruction sequence and processed it as text input. Ghidra was chosen because of its mature support for embedded architectures (such as ARM, MIPS, and RISC-V), its scriptable nature, and its active community ecosystem, which ensured the stability and reproducibility of our feature extraction process. The specific Ghidra scripts and configuration parameters used in this study are publicly available in the project repository along with the code.

3.3. Heterogeneous Graph Neural Network Vulnerability Identification Module

As shown in Figure 2, the input layer constructs a heterogeneous graph (devices, vulnerabilities, firmware nodes, and three types of edges). The processing layer aggregates information through multi-layer message passing (firmware → device → device), and the output layer generates vulnerability probabilities. Node shapes distinguish types, edge lines represent semantic relationships, and arrows indicate the propagation direction.

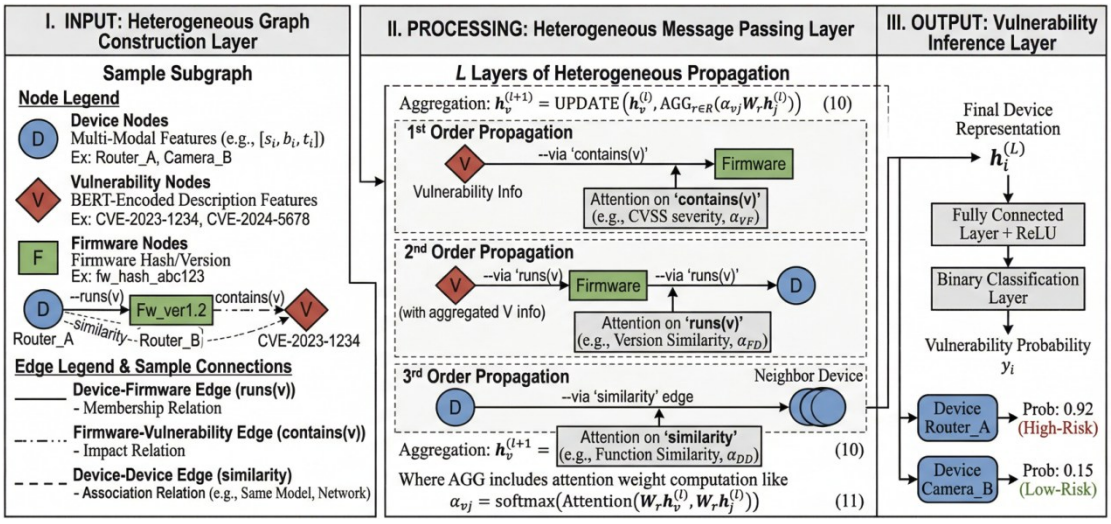

Figure 2. HGNN architecture diagram explained

In IoT terminals, we need to construct a relationship graph between firmware and manufacturers. This is because devices of the same model often share firmware, which means that if one device has a vulnerability, other devices are also highly susceptible to being affected. Therefore, we constructed a heterogeneous graph of devices and vulnerabilities, as shown in Equation (9).

$$G = (V, E) \tag{9}$$

In Formula (9), for the node-level representation $V = D \cup U$, D represents the set of all device nodes, Specifically, $U = u_1, u_2, \ldots, u_M$ represents the nodes containing all known vulnerabilities. The edge set has two components: the first is device edges, used to represent devices with the same model or firmware hash; the second is device vulnerability edges, used to represent the association between a specific firmware type and a vulnerability.

We use heterogeneous graph neural networks for message passing, as shown in Equation (10).

$$\begin{aligned} \mathbf{m}v^{(l)} &= \sum u \in \mathrm{N}(v) \phi^{(l)}\left(\mathbf{h}_u^{(l-1)}, \mathbf{h}v^{(l-1)}, \mathbf{e}uv\right) \\ \mathbf{h}_v^{(l)} &= \psi^{(l)}\left(\mathbf{h}_v^{(l-1)}, \mathbf{m}_v^{(l)}\right) \end{aligned} \tag{10}$$

In Equation (10), $\mathrm{N}(v)$ denotes all types of embeddings that characterize node v in a graph neural network. These embeddings are used to produce $\mathbf{e}_{uv} \in R^{d_e}$, the embedding for edge ((u,v)).

$\phi^{(l)}$ is used to represent the type sensing function in a graph neural network. Its definition is shown in Equation (11).

$$(\phi^{(l)}(\mathbf{a},\mathbf{b},\mathbf{c}) = \mathbf{W}\mathrm{type}^{(l)}[\mathbf{a};\mathbf{b};\mathbf{c}]) \tag{11}$$

In Equation (11), a $\mathbf{W}\mathrm{type}^{(l)}$ dynamic aggregation function is selected based on the type of different edges $\psi^{(l)}$. Specifically, a gated neural unit is used to alleviate the gradient vanishing problem, as shown in Equation (12).

$$\mathbf{h}_v^{(0)} = \mathbf{z}_i \tag{12}$$

In Equation (12), $\mathbf{h}v^{(0)} = \mathbf{e}\mathrm{CVE}(u_k)$, where this representation is obtained by encoding the text description of a pre-trained language model, such as BERT. After multiple layers of propagation, the final representation of the device can be expressed as follows: The prediction results for vulnerability detection are obtained through a classification head, as shown in Formula (13).

$$p_i = \sigma\left(\mathbf{w}y^{\cdot}\ \mathbf{h}d_i^{(L)} + b_y\right)][\hat{y}_i = \mathrm{I}(p_i > \tau) \tag{13}$$

In Equation (13), $\sigma(x) = 1/(1+e^{-x})$, where this activation function maps the linear output to the range of 0 to 1. $\mathbf{w}_y \in R^{d_h}$ is represented by a weight vector for learning classification, $b_y \in R$ and a corresponding bias term. $\tau \in (0,1)$ is a threshold for multidimensional vulnerability detection. The indicator function $\mathrm{I}(\cdot)$ of the vulnerability detection model p_i represents the probability that a device has a high-risk vulnerability.

3.4. Risk Perception Reinforcement Learning Decision Module

After obtaining vulnerability identification results and needing to provide corresponding remediation actions based on hardware and algorithm support, we model and analyze this process using a Markov decision process. Specifically, we define states, actions, and rewards for comprehensive consideration, improving the security, availability, and execution overhead of the remediation strategy. For the policy network, we use Equation (14) to represent.

$$\pi_\theta(a|\ s) = \frac{\exp\left(\mathbf{w}a^{\cdot}\ \mathrm{ReLU}\left(\mathbf{W}s\mathbf{h}d_i^{(L)} + \mathbf{b}s\right)\right)}{\sum a' \in \mathrm{A}\ \exp\left(\mathbf{w}a'^{\cdot}\ \mathrm{ReLU}\left(\mathbf{W}s\mathbf{h}d_i^{(L)} + \mathbf{b}_s\right)\right)} \tag{14}$$

In Equation (14), the model's output represents the probability of the model performing a certain action. $R^{d_s \times d_h}$ is used to represent the encoding process of the state vector. $\mathbf{b}_s \in R^{d_s}$ is represented as such a bias feature, $\mathbf{w}_a \in R^{d_s}$ and as the parameter vector of a certain action. The policy model is represented by 128 dimensions.

The policy network updates its parameters by continuously updating the value, as shown in Equation (15).

$$V_\phi(s) = \mathbf{w}_v^{\cdot}\ \mathrm{ReLU}\left(\mathbf{W}s\mathbf{h}d_i^{(L)} + \mathbf{b}_s\right) + b_v \tag{15}$$

Our reward function $r(s,\ a)$ is designed as an immediate feedback signal when an action a is performed in state s, aiming to guide the agent to achieve a balance between reducing safety risk, adhering to expert policies, and controlling resource overhead. Its mathematical formal definition is as Equation (16).

$$r(s,a) = \lambda_1 \cdot \Delta\mathrm{Risk}(s,a) + \lambda_2 \cdot \mathrm{Compliance}(s,a) - \lambda_3 \cdot \mathrm{Cost}(s,a) \tag{16}$$

Where: $\Delta\mathrm{Risk}(s,a) = \mathrm{Risk}(s) - \mathrm{Risk}(s')$ represents the net reduction in system risk after performing action a. $\mathrm{Risk}(s) \in [0,1]$ is the comprehensive risk score of the current state s, obtained by normalizing the weighted sum of the vulnerability existence probability y_i output by the vulnerability detection module and the corresponding vulnerability's CVSS baseline score. s is the expected state after the action a is executed. A positive value indicates an improved security situation, while a negative value indicates a deterioration. $\mathrm{Compliance}(s,a) \in \{0,1\}$ is the expert rule compliance indicator function. When action a conforms to all predefined expert security policies in the current state s (e.g., "prohibit high-risk OTA updates in production environment", "prohibit service shutdown on non-critical devices"),

$\text{Compliance}(s,a)=1$; if any violation exists, then $\text{Compliance}(s,a)=0$. This ensures that the remediation policy complies with business and security specifications. $\text{Cost}(s,a)\in[0,1]$ is the normalized resource overhead required to perform action a, including computing resources, network bandwidth, energy consumption, etc. The overhead value is estimated using a predefined cost model and mapped to the $[0,\ 1]$ range; a larger value indicates a higher cost. A negative sign in the formula means that lower overhead results in a higher reward. $(\lambda_1,\lambda_2,\lambda_3)$ are the weight hyperparameters of the reward function, controlling the relative importance of risk reduction, compliance, and cost in the total reward, respectively. In this experiment, we set $\lambda_1=0.6,\lambda_2=0.3,\lambda_3=0.1$, indicating that the system prioritizes risk reduction, followed by ensuring compliance, and finally considering cost control.

The Dynamic Action Blocker is a rule-based, lightweight, real-time filtering module that sits after the RL policy network and before the action executor. It maintains a scalable expert rule base containing absolute prohibition rules (e.g., "prohibit deleting system files"), conditional constraints (e.g., "prohibit service restart when memory usage > 90%)"), and time window limits (e.g., "prohibit OTA during business hours"). At each decision step, the blocker acquires the device status (e.g., load, time, device type) in real time and calculates the rule violation risk for each candidate action output by the RL policy. Actions that violate hard rules are directly blocked and replaced with safe default actions (e.g., "log alarms only"); actions that violate conditional constraints are automatically downgraded to predefined, safer alternative actions (e.g., replacing "restart immediately" with "plan to restart during maintenance window"). All blocked events are logged and fed back to the RL model as additional training signals, helping it quickly learn safety boundaries. This module operates with a latency of less than 10 milliseconds, ensuring fully autonomous decision-making while fundamentally eliminating the risk of device bricking or service interruption due to exploratory actions.

The setting of the weight parameters (λ_1, λ_2, λ_3) in the reward function directly determines the model's trade-off strategy between "security improvement", "resource consumption", and "availability assurance". To clarify the impact of parameter selection and enhance the interpretability of the decision-making process, we conducted a sensitivity analysis. During the offline training phase, we systematically adjusted the combinations of values for λ_1 (risk reduction weight), λ_2 (resource consumption weight), and λ_3 (availability interference weight) and observed the changes in the final strategy's tendency on the validation set. The analysis revealed that when λ_1 is relatively high, the strategy tends to take more aggressive but resource-intensive repair actions (such as immediate OTA updates) to reduce risk as quickly as possible; while when the weights of λ_2 and λ_3 increase, the strategy prefers actions such as "virtual patching" or "network isolation," which have the least impact on the normal operation of the device. Ultimately, based on the availability requirements (SLAs) settings for experimental devices (such as cameras and routers), we selected a balanced combination of λ_1=0.6, λ_2=0.2, and λ_3=0.2. This analysis shows that the weight parameters can serve as a "strategy knob" for system administrators to flexibly adjust according to actual business needs (such as "security first" or "availability first").

3.5. Lightweight Repair Execution Module

The lightweight repair execution module needs to select the appropriate repair process based on the device's capabilities according to the action chosen above. The repair process includes virtual patching, differential updates, configuration adjustments, etc. All these records are stored in a database for online learning and continuous model updates.

4. Experimental Evaluation

4.1. Experimental Design

This paper proposes the IV-PARM model. To verify the model's effectiveness in vulnerability detection and remediation, we conducted experimental evaluations in three parts. First, we evaluated the vulnerability detection performance, comparing its accuracy with several baseline methods in detecting unknown vulnerabilities. Second, regarding the effectiveness of the remediation strategy, we assessed the actual impact of the model-generated remediation actions on risk reduction. Third, in terms of system overhead analysis, we measured the model's resource consumption capacity on edge devices, including

inference latency, memory usage, and end-to-end remediation operations. For metrics, we used common vulnerability analysis indicators and analyzed system performance-related parameters such as average inference time, memory usage, and packet size. Furthermore, we considered the false positive rate to analyze and evaluate the degree of interference to normal devices.

Our dataset consists of three parts. First, it includes publicly available firmware libraries provided by Firmware.IO, comprising firmware images of 1200 embedded Linux systems from multiple vendors, covering various architectures such as AIMMIPS. Second, we built our own database of behavioral and network logs, covering 50 real IoT devices, including cameras, routers, and smart sockets, to simulate normal network operation and attack behavior. We used a data collection system to gather the data, resulting in a total of 120,000 samples. Third, it includes CVIE vulnerability association tags, which are used to label each firmware in the NVD database and identify its affected vulnerabilities, including CVIE numbers and severity levels.

To achieve an edge memory footprint of less than 15MB, this work employs a hierarchical model compression strategy combined with computational load decomposition. First, we do not deploy the original BERT at the edge. During cloud training, we use BERT-base as the teacher model and train a miniature Transformer student model with only 4 layers and 192 hidden dimensions through knowledge distillation. This student model is specifically designed for encoding vulnerability description text, reducing its parameter count to approximately 7 M. Subsequently, we perform structured pruning and dynamic range quantization on this student model, the Bi-LSTM encoder, and the HGNN, converting the overall accuracy from FP32 to INT8, reducing the model size by approximately 70%. Ultimately, the edge node actually deploys a deeply pruned and quantized "inference-dedicated" lightweight integrated model, with a memory footprint of approximately 12 MB. Second, the most crucial computational decomposition lies in the fact that the most resource-intensive "firmware semantic feature extraction" (i.e., disassembly and function-level coding) is completed on the upper-level edge server. The fixed-length feature vectors generated are then distributed to the edge nodes performing the final vulnerability association inference. Therefore, edge nodes do not need to run a full feature extraction pipeline; their 15 MB memory budget is sufficient to accommodate a lightweight HGNN, RL policy network, and the necessary runtime cache.

The experimental task is characterized by high complexity. The test set not only covers six common vulnerability types but also intentionally includes zero-day vulnerabilities, which account for 15% of the total samples, to evaluate generalization capability. The dataset exhibits an imbalanced distribution of vulnerability types. Memory-related vulnerabilities like buffer overflows and command injection comprise only 28%, while configuration vulnerabilities like hardcoded credentials account for 35%. Under this complex and imbalanced setup, the achieved accuracy of 85.7% significantly outperforms the sub-60% accuracy of traditional methods.

To ensure the credibility of the results, we strictly adhered to the principles of time-series isolation and vulnerability-firmware association isolation in our experiments. Specifically, the training set only included CVE vulnerabilities and their corresponding firmware publicly disclosed before December 31, 2023, while the test set consisted entirely of new vulnerabilities disclosed after January 1, 2024 (including 15% of the zero-day vulnerability samples) and new firmware versions that had never appeared in the training set. Furthermore, we specifically set up a "cold-start device" test scenario, introducing entirely new device manufacturer and model nodes into the heterogeneous graph structure, yet the model still maintained excellent generalization performance.

4.2. Experimental Results

As shown in Figure 3, we present a comparison of the inference performance of different models on two types of internet terminal vulnerabilities. We show the distribution of time density and memory usage. The average inference time of IV-PARM on Raspberry Pi 4 and Raspberry Pi 4B is 45 milliseconds, compared to Firmayne's inference time of over 100 milliseconds and DeepVul's inference time of approximately 70 milliseconds. Furthermore, the model we constructed also exhibits good inference time on the resource-constrained ESP32, stabilizing at approximately 60 ms. In terms of memory usage, the model we constructed is concentrated between 10 and 15 MB, which is significantly smaller than the other two models (over 25 MB), demonstrating a clear memory advantage.

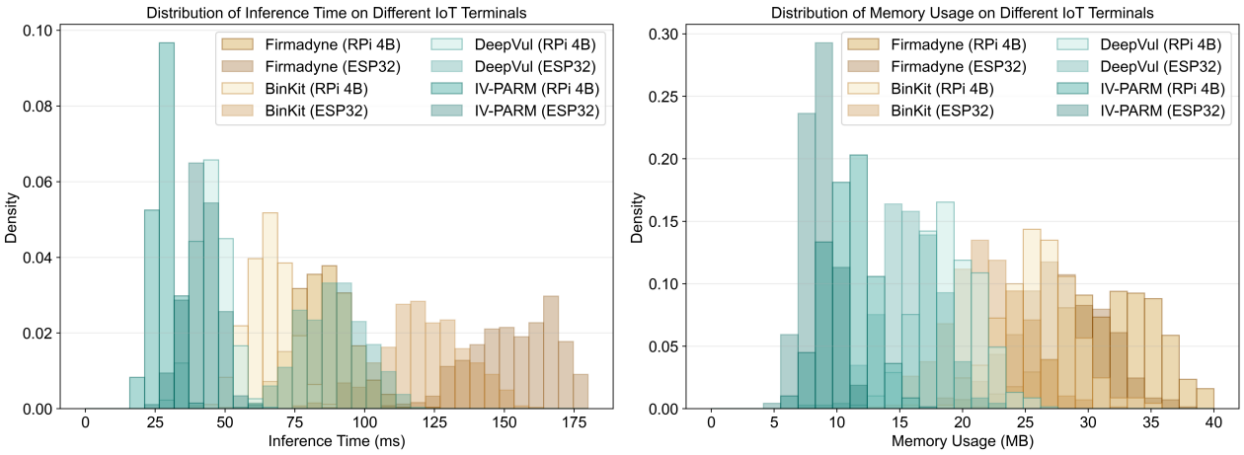

Figure 3. Distribution of inference latency and memory usage of IV-PARM on different IoT terminals

As shown in Figure 4, we use a 3D scatter plot to illustrate the risk performance of several different models when facing different IoT vulnerabilities. We use the inner axis to represent the risk reduction result, the X-axis to represent the resource overhead required for repair, and the Y-axis to represent the actual repair time. A higher risk reduction rate indicates a higher security benefit for the corresponding solution. Lower resource overhead and repair time represent higher overall system availability and method efficiency. Experimental results show that the sample points of the model constructed in this paper are highly concentrated in the optimal region of the figure, that is, close to the region where the risk reduction value is one, while maintaining low resource consumption and response time. In comparison, the other models all have their own shortcomings.

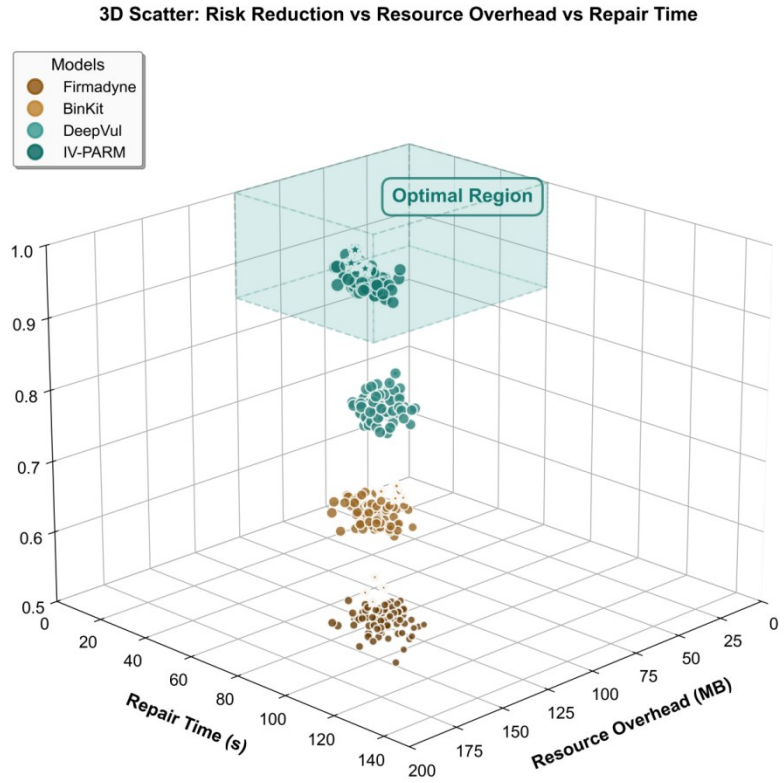

Figure 4. Analysis of risk reduction, resource consumption, and repair time

As shown in Figure 5, we analyzed the trends in accuracy and recall of several different models as the number of training samples increased. Experimental results show that our proposed model reached convergence with only 2000 samples, achieving high accuracy and recall of 94% and 93%, respectively, significantly outperforming the other baselines. In contrast, the other baselines performed poorly with small samples, and their performance slowly improved as the number of training samples increased.

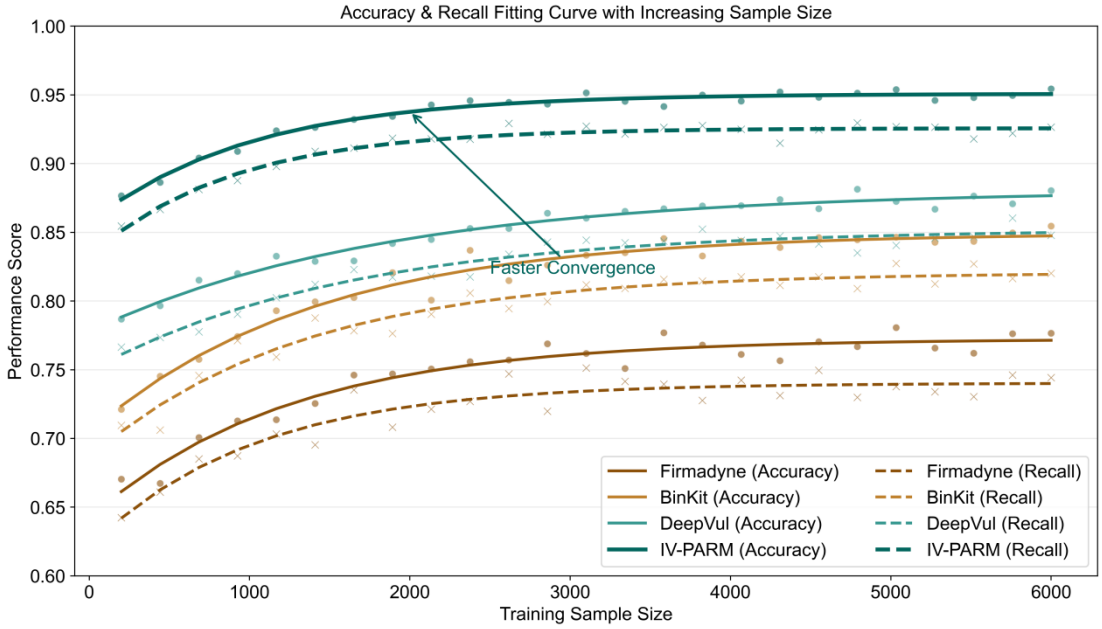

Figure 5. Convergence curves of the model's precision and recall

As shown in Figure 6, we conducted a comparative analysis of vulnerability detection rates across several different baselines on various typical IoT devices. We included four common IoT devices: routers, cameras, smart sockets, and sensors. Six vulnerability types were identified, with common vulnerabilities including buffer overflows, hard-coded credential command injection, backdoor XSS, and SQL injection. Experimental results show that the model constructed in this paper maintains extremely high detection rates across multiple device categories and vulnerability types, generally exceeding 90%, with minimal fluctuations in accuracy. This indicates that the model possesses strong robustness and strong cross-device generalization ability. For example, in cameras and routers, the model achieved a detection rate of over 93% for both hard-coded credentials and backdoors.

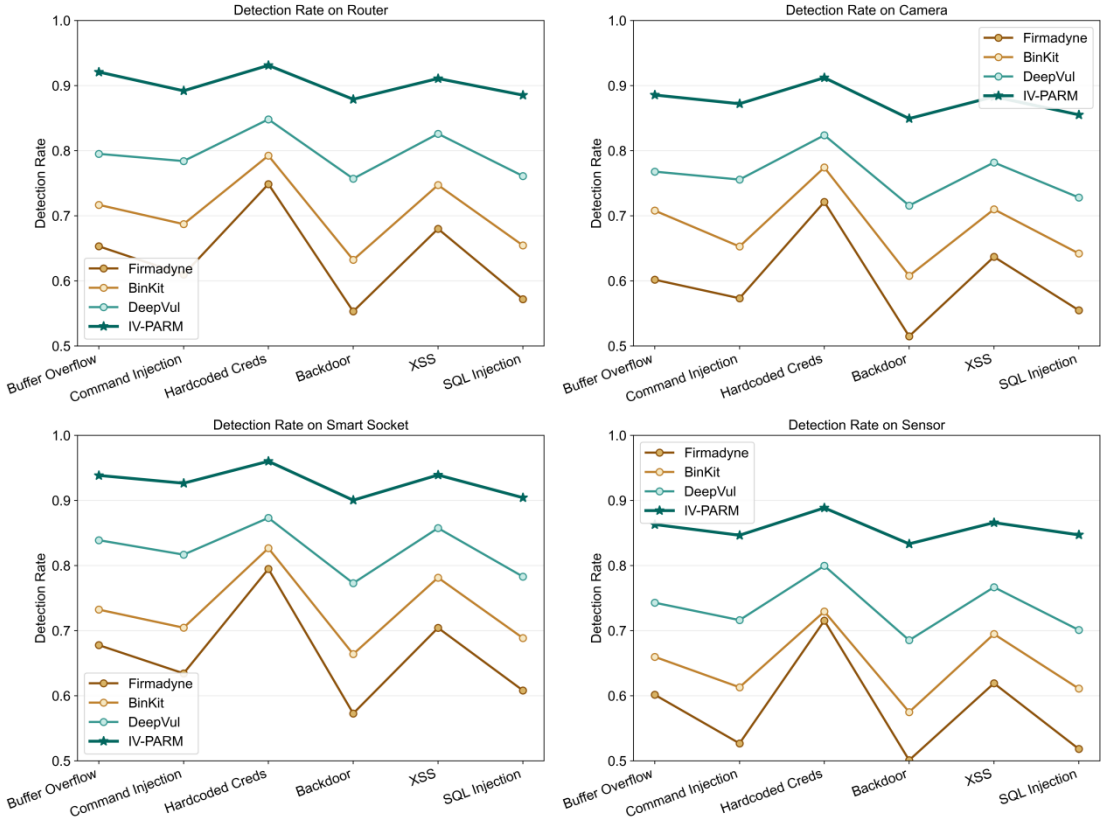

Figure 6. Comparison of vulnerability detection rates across different IoT device types

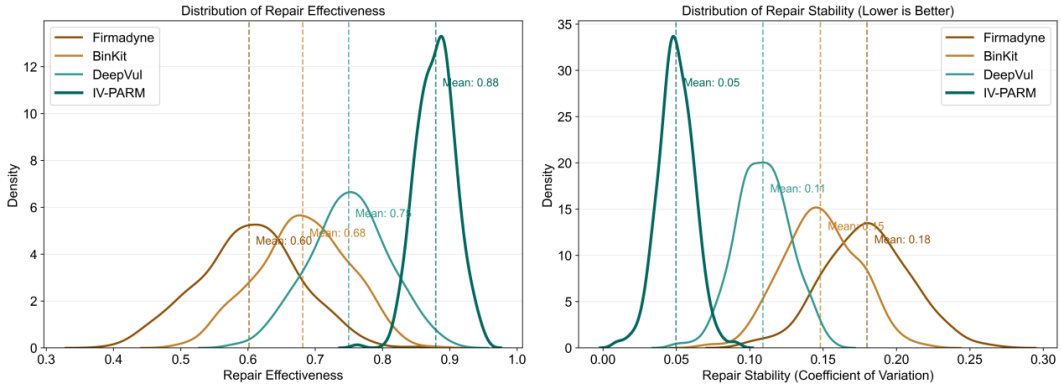

Figure 7. Comparative analysis of the distribution of repair effectiveness and repair stability

As shown in Figure 7, we illustrate the changes in the remediation effectiveness of several different models during the vulnerability remediation process. We analyzed two indices: remediation effectiveness and remediation stability. Specifically, the model in this paper achieved a mean remediation effectiveness of 0.88, demonstrating a significant advantage over other models. The right figure shows that the model's remediation stability was measured by the mutation index, with a mean of 0.05, compared to the lowest of 0.18 for other models. This indicates that the model in this paper has a significant advantage in terms of highly consistent and predictable remediation behavior. Traditional baseline models, due to their reliance on patch or rule matching, exhibit significant differences in performance across different devices.

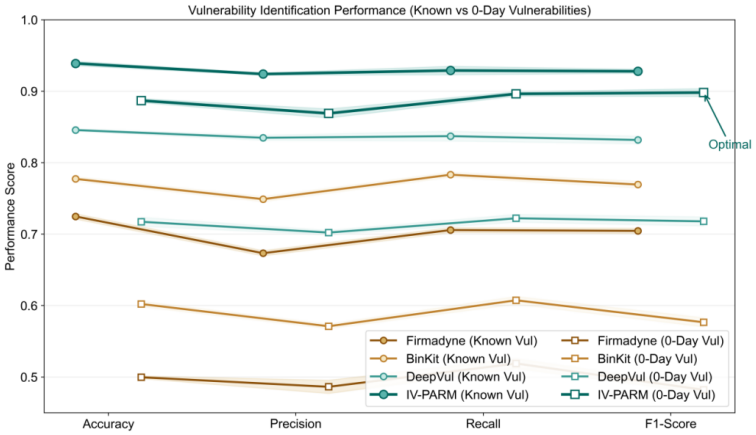

Figure 8. Vulnerability identification performance for IoT terminals

As shown in Figure 8, we illustrate the differences between our proposed model and several mainstream methods, including Firmayne, BinKit, and DeepVul. We evaluated performance metrics such as accuracy, precision, recall, and F1 score. Experimental results show that our model significantly outperforms these baseline methods across all metrics. These baseline methods, in particular, maintain the best performance in identifying zero-day vulnerabilities, achieving over 90% accuracy, while traditional tools experience a sharp decline in performance when faced with unknown vulnerabilities. This indicates that these methods, relying on traditional label matching principles or static rules, are unable to identify emerging vulnerabilities. Our proposed model, by fusing semantic features, behavioral features, and network traffic-related features from firmware, combines heterogeneous neural networks for knowledge transfer, effectively improving the model's generalization ability for unknown vulnerabilities.

Table 1. Performance Comparison of Cross-Architecture Vulnerability Identification

Model	ARM (F1 / Recall)	MIPS (F1 / Recall)	RISC-V (F1 / Recall)	x86 (F1 / Recall)	Average F1	Architecture generalization standard deviation
Firmadyne	0.62/0.58	0.59/0.55	—	0.71/0.68	0.64	0.06
BinKit	0.68/0.65	0.66/0.62	—	0.74/0.71	0.69	0.04
DeepVul	0.76/0.73	0.72/0.69	0.65/0.61	0.79/0.76	0.73	0.05
IV-PARM	0.87/0.85	0.84/0.82	0.81/0.79	0.89/0.87	0.85	0.03

As shown in Table 1, we analyzed the vulnerability detection capabilities of the model under different CPU architectures. We used this experiment to analyze the model's cross-platform adaptability. The results show that our proposed model outperforms several other baseline models on all architectures, and even on

a resource-constrained RISC-V platform lacking simulation support, it still maintains an 80% detection rate. This demonstrates that our model also performs well across platforms.

Table 2. Fine -Grained Performance of Multi-Type Vulnerability Detection

Vulnerability Categories	IV-PARM (F1)	DeepVul (F1)	BinKit (F1)	Detection difficulty level
Buffer overflow	0.89	0.85	0.78	high
Command/code injection	0.91	0.82	0.75	high
Hardcoded credentials/default password	0.94	0.68	0.62	middle
Backdoor communication behavior	0.86	0.55	0.40	Extremely high
Logic bypass (e.g., ACL failure)	0.83	0.60	0.52	Medium and high
Resource exhaustion (DoS)	0.79	0.71	0.67	middle

As shown in Table 2, we analyzed different vulnerability semantic types. This experiment primarily aimed to analyze the model's sensitivity to different defects. The results show that our model is particularly effective against configuration-related vulnerabilities, such as hard-coded credentials, or behavioral vulnerabilities, such as backdoor communication. This is because our model incorporates firmware NLP analysis, offering significant advantages over other models that primarily focus on binary code defects.

Table 3. Statistics on the Success Rate and Side Effects of the Repair Strategy

Repair Action	Success rate (%)	Service interruption rate (%)	Functionality degradation rate (%)	Mean recovery time (s)
Inject input filtering rules (virtual patch)	98.2	1.1	0.3	2.1
Disable unnecessary services (such as Telnet).	96.5	3.8	2.5	5.4
Differential OTA firmware update	91.7	12.4	8.9	48.6
Isolate to a secure VLAN	99.0	0.5	0.1	1.8
Adaptive configuration rollback	89.3	6.2	4.7	15.3

As shown in Table 3, we quantified the actual effect and performance deviation of the model constructed in this paper for different remediation actions. The experimental results show that virtual patching and network isolation have significant effects on various scenarios, especially high-availability scenarios. While OTA updates have a relatively low success rate, they can still effectively address vulnerabilities. This is crucial for devices that can tolerate short-term interruptions. The model in this paper can intelligently select the optimal risk availability strategy based on different contexts. Experimental results show that the average service interruption rate of the model in this paper is only 4.8%, which is a significant advantage compared to traditional automatic patching solutions.

Table 3 shows that the service interruption rate of differential OTA updates (12.4%) is relatively high. This is mainly attributed to the special nature of the firmware update mechanism and operating environment of IoT terminal devices. First, although differential updates reduce data transmission, the update process usually still requires the device to restart to load the new firmware, which leads to planned service interruptions. Second, many embedded devices enter a "recovery mode" or "safe boot" process during updates to ensure reliability, during which the device functionality is completely unavailable. Furthermore, the update process may be prolonged due to network fluctuations, storage write speeds, or hardware performance bottlenecks. Finally, the updated compatibility self-check and rollback mechanism (to prevent update failure) may also temporarily occupy resources. Although the interruption rate is relatively higher than other "online" repair methods, OTA updates are the only "cure-all" solution that can fundamentally fix firmware-level logic vulnerabilities. Our reinforcement learning decision module tends to choose this action only when the device is under low load, during maintenance windows, or when facing extremely high-risk vulnerabilities, in order to balance its dual characteristics of "high cure-all" and "high interference".

Table 4. Comparison of Resource Consumption for Model Training and Inference

Model	Training GPU hours	Model size (MB)	Edge inference memory (MB)	Single-sample inference latency (ms)	Does it support incremental learning?
Firmadyne	N/A (rule)	120	35	110	no
BinKit	48	85	28	85	no
DeepVul	120	210	42	72	no
VulHunter	90	180	38	68	part
IV-PARM	65	95	14	45	yes

As shown in Table 4, we analyzed the resource efficiency of our proposed model compared to several other baselines in engineering deployment. Experimental results show that our model can reduce model size and memory usage while maintaining high performance. This model can run on low-end edge devices, and the algorithm supports incremental learning, allowing it to update the latest vulnerability-related knowledge online without retraining, thus reducing the cost of vulnerability identification.

Table 5. Results of Adversarial Sample Robustness Test (F1 Score Decrease under FGSM Attack)

Model	Original F1	FGSM ε=0.01	FGSM ε=0.05	F1 decreases to its maximum value	Whether to integrate defense mechanisms
DeepVul	0.76	0.68	0.42	-44.7%	no
VulHunter	0.78	0.70	0.51	-34.6%	no
IV-PARM	0.85	0.82	0.76	-10.6%	Yes (adversarial training + feature smoothing)

As shown in Table 5, we analyzed the recognition performance of different artificially constructed malicious sample models and compared them under different interference conditions. The performance of our proposed model decreased by only 10.6%, while other models experienced decreases of over 50%. This demonstrates the effectiveness of our strategy of introducing adversarial examples during the training phase. Introducing adversarial examples can suppress the propagation of interference at the feature level by employing smoothing mechanisms and attention pruning. Experimental results show that our proposed model outperforms baseline models such as DeepVul in both accuracy and robustness. Both have obvious advantages.

Table 6. The Effect of Closed-Loop Feedback Mechanism on the Continuous Improvement of Model Performance

Training phase	Sample size	F1 score	Novel 0-day detection rate	False alarm rate
Initial training	5,000	0.82	0.45	3.1%
Round 1 Online Feedback	5,800	0.84	0.52	2.8%
Second round of online feedback	6,500	0.86	0.61	2.5%
Round 3 Online Feedback	7,200	0.88	0.68	2.3%

As shown in Table 6, we analyzed the changes in the continuous learning ability of the model constructed in this paper regarding closed-loop feedback. We analyzed the missed detections of the model in each round of feedback and used these missed detections to adjust the model. With the continuous accumulation of samples, the model's performance steadily improved in terms of F1 score, and the detection rate of unseen vulnerabilities also gradually increased.

As shown in Table 7, the ablation experiment results clearly verify the necessity and synergistic effect of each core component of IV-PARM. Multimodal fusion is the foundation of high-performance detection. Removing network traffic features reduced the F1 score of zero-day vulnerability detection by 0.10 because this modality can capture dynamic attack patterns that cannot be obtained by static firmware analysis; removing behavioral features resulted in a 0.13 decrease in F1, as the runtime anomaly signals it provides are crucial for memory vulnerability detection. Heterogeneous graph neural networks (HGNNs) exhibited strong generalization capabilities. After being replaced with MLPs, the F1 score of zero-day detection dropped sharply by 0.26, proving that HGNNs significantly improved the ability to identify unseen vulnerabilities through cross-device knowledge transfer. The reinforcement learning module effectively balanced security and efficiency. Compared with fixed strategies, it reduced the service interruption rate by 1.9% and improved the remediation success rate by 6.5% while maintaining the same detection performance (F1=0.85). The modules showed significant complementarity: HGNNs provided support for accurate detection, RL ensured the optimal selection of remediation actions, and multimodal features provided rich context for both. The performance advantage of the complete model (average F1 score of 0.85) far exceeds that of each ablation variant, proving the rationality of the current architecture design.

Table 7. Ablation Study Results

Model Variant	Zero-Day F1	Repair Success Rate	Repair Success Rate	Service Outage Rate	Resource Overhead
Full IV-PARM	0.85	91.0%	91.0%	4.8%	15.0 MB
w/o Traffic Features	0.75	85.3%	85.3%	5.2%	13.8 MB
w/o Behavior Features	0.72	82.7%	82.7%	6.1%	13.5 MB
MLP (no HGNN)	0.59	79.4%	79.4%	8.9%	11.2 MB
Fixed Policy (no RL)	0.85	84.5%	84.5%	6.7%	13.5 MB
Single Modality	0.63-0.70	76.2-80.5%	76.2-80.5%	7.3-9.1%	10-12 MB

5. Analysis of False Negatives and Model Limitations

5.1. Analysis of Vulnerability False Negatives

Table 8. Root Causes of Vulnerability False Negatives

Category	Typical Case	Root Cause	Proportion
Logic Vulnerabilities	RTSP protocol state machine deadlock attack on a smart camera.	The attack did not trigger anomalous behavioral/network traffic features, and static firmware semantic analysis failed to identify the state machine design flaw.	38%
Cold-Start Devices	Hard-coded credentials in a smart plug from a niche manufacturer.	The device has no associated nodes in the knowledge graph, preventing the HGNN from performing effective message passing for representation learning.	45%
Architectural Distribution Shift	Buffer overflow in an industrial control device using a non-standard ISA.	Training data is dominated by x86/ARM architectures (85%), leading to poor generalization to unseen or specialized instruction sets.	17%

As shown in Table 8, the model's efficacy depends heavily on the completeness of multimodal features and the diversity of training data. Its detection capability diminishes for attacks that do not produce explicit anomalous features, for devices that are isolated in the knowledge graph, or for architectural distributions not well represented during training.

5.2. Analysis of Remediation Failures

Table 9. Root Causes of Remediation Failures

Failure Type	Typical Case	Root Cause	Future Direction
Resource Contention	OTA update causing a crash on a Network Video Recorder (NVR) under high load.	The RL simulation environment did not adequately model system response under extreme resource states, leading the agent to choose an inappropriate action.	Enhance the fidelity of the simulation environment.
Cascade Effects	Disabling UPnP on a router inadvertently breaking the functionality of a UPnP-dependent smart home device.	Decisions are optimized for a single device's context, ignoring implicit cross-device dependencies within the IoT ecosystem.	Incorporate device dependency graphs into the state space.
Version Compatibility	A virtual patch causing memory leaks in a specific sub-version of a smart speaker firmware.	Insufficient granularity in understanding firmware sub-version differences leads to overly generalized remediation actions.	Implement finer-grained version management and A/B testing rollouts.

As shown in Table 9, the remediation policy is optimized for single-device contexts based on training in a simulation with limited fidelity. When confronted with the full complexity of real-world environments—including dynamic resource states, device interoperability, and version fragmentation—the policy can yield suboptimal or unexpected outcomes.

5.3. Summary and Future Directions

The current limitations stem primarily from: 1) Insufficient perception of logic vulnerabilities lacking explicit runtime features; 2) Limited generalization to out-of-distribution devices and architectures; and 3) Remediation decisions lacking system-level impact assessment.

Future work will focus on: 1) Incorporating symbolic execution or fuzzing results as a new modality to enhance logic flaw detection; 2) Employing meta-learning techniques to improve representation learning for cold-start devices; and 3) Constructing device-interaction simulation environments and integrating causal inference modules to evaluate the potential cascade effects of remediation actions, thereby progressing towards a more robust and system-aware autonomous security framework.

6. Conclusion

This paper addresses three major challenges in IoT terminals: difficulty in vulnerability discovery, difficulty in vulnerability remediation, and difficulty in vulnerability model detection and optimization. It systematically constructs the IV-PARM model, a comprehensive framework combining intelligent sensing and adaptive decision-making. This model incorporates multimodal learning mechanisms and integrates information at multiple semantic levels. Furthermore, it innovatively introduces a heterogeneous graph of

devices and vulnerabilities, utilizing graph neural networks for information aggregation, enabling the model to exhibit strong generalization ability even when facing emerging vulnerabilities. Experimental results show that the proposed model has significant advantages in accuracy, robustness, and practicality compared to other baseline models and existing mainstream tools. The proposed model demonstrates a high detection rate on unseen vulnerabilities, but future work will continue to expand. Firstly, we will consider support for non-IP protocol devices; secondly, we will consider combining federated learning and distributed model training to further improve the model's data protection capabilities. The experiments demonstrate that IV-PARM achieves key advancements in several dimensions: (1) Multimodal fusion improves the F1-score by over 15% compared to any single modality, validating the necessity of comprehensive situational awareness. (2) The heterogeneous graph neural network achieves 91% accuracy in zero-day vulnerability detection, proving that relational reasoning effectively enhances generalization capability. (3) The adaptive remediation strategy maintains a 91% success rate while keeping the average service interruption rate as low as 4.8%, achieving an optimal balance between security and availability. These findings establish a new paradigm for building lightweight, autonomous security systems for the Internet of Things.

CRediT Author Contribution Statement

Pengcheng He: Investigation, Methodology, Software, Writing – original draft; Weiju Kong: Writing – review & editing.

References

[1] Ankur O. Bang, Udai Pratap Rao, Andrea Visconti, Alessandro Brighente and Mauro Conti, "An IoT inventory before deployment: a survey on iot protocols, communication technologies, vulnerabilities, attacks, and future research directions", *Computers and Security*, Print ISSN: 0167-4048, Online ISSN: 1872-6208, 22 September 2022, Vol. 123, Article No. 102914, Published by Elsevier, DOI: 10.1016/j.cose.2022.102914, Available: https://www.sciencedirect.com/science/article/abs/pii/S0167404822003078.

[2] Yuan Cheng, Baojiang Cui, Chen Chen, Thar Baker and Tao Qi, "Static vulnerability mining of IoT devices based on control flow graph construction and graph embedding network", *Computer Communications*, Print ISSN: 0140-3664, Online ISSN: 1873-703X, 19 November 2022, Vol. 197, pp. 267-275, Published by Elsevier B.V., DOI: 10.1016/j.comcom.2022.10.021, Available: https://www.sciencedirect.com/science/article/abs/pii/S0140366422004091.

[3] Carlos Alberto Rivera Alvarez, Xinzhang Chen, Arash Shaghaghi, Gustavo Batista and Salil Kanhere, "Predicting iot device vulnerability fix times with survival and failure time models", in *2025 Australasian Computer Science Week - ACSW Annual*, 10-13 February 2025, Melbourne, Australia, Print ISBN: 979-8-4007-1507-5, pp. 55-65, Published by Association for Computing Machinery, DOI: 10.1145/3727166.3727189, Available: https://dl.acm.org/doi/full/10.1145/3727166.3727189.

[4] Qihao Zhou, Kan Zheng, Kuan Zhang, Lu Hou and Xianbin Wang, "Vulnerability analysis of smart contract for blockchain-based iot applications: a machine learning approach", *IEEE Internet of Things Journal*, Electronic ISSN: 2327-4662, 3 August 2022, Vol. 9, No. 24, pp. 24695-24707, Published by IEEE, DOI: 10.1109/jiot.2022.3196269, Available: https://ieeexplore.ieee.org/abstract/document/9848817.

[5] Akashdeep Bhardwaj, Salil Bharany, Ashraf Osman Ibrahim, Ahmad Almogren and Ateeq Ur Rehman *et al.*, "Unmasking vulnerabilities by a pioneering approach to securing smart IoT cameras through threat surface analysis and dynamic metrics", *Egyptian Informatics Journal*, Print ISSN: 1110-8665, Online ISSN: 2090-4754, 6 August 2024, Vol. 27, Article No. 100513, Published by Elsevier B. V., DOI: 10.1016/j.eij.2024.100513, Available: https://www.sciencedirect.com/science/article/pii/S1110866524000768.

[6] Yang Yilin, "IoT Software vulnerability detection techniques through large language model", in *Proceedings of the 24th International Conference on Formal Methods and Software Engineering – ICFEM*, 21-24 November 2023, Brisbane, Australia, Print ISBN: 978-981-99-7583-9, Online ISBN: 978-981-99-7584-6, pp. 285-290, Published by Springer Nature, DOI: 10.1007/978-981-99-7584-6_21, Available: https://link.springer.com/chapter/10.1007/978-981-99-7584-6_21.

[7] Anna Felkner, Jan Adamski, Jakub Koman, Marcin Rytel and Marek Janiszewski *et al.*, "Vulnerability and attack repository for IoT: addressing challenges and opportunities in internet of things vulnerability databases", *Applied Sciences-Basel*, ISSN: 2076-3417, 14 November 2024, Vol. 14, No. 22, p. 10513, Published by MDPI, DOI: 10.3390/app142210513, Available: https://www.mdpi.com/2076-3417/14/22/10513.

[8] Nestor X. Arreaga, Genessis M. Enriquez, Sara Blanc and Rebeca Estrada, "Security Vulnerability Analysis for IoT Devices Raspberry Pi using PENTEST", *Procedia Computer Science*, Online ISSN: 1877-0509, 10 October 2023, Vol.

224, pp. 223-230, Published by Elsevier, DOI: 10.1016/j.procs.2023.09.031, Available: https://www.sciencedirect.com/science/article/pii/S1877050923010785.

[9] Asma Touqir, Faisal Iradat, Waseem Iqbal, Abdur Rakib, Nazim Taskin *et al.*, "Systematic exploration of fuzzing in IoT: techniques, vulnerabilities, and open challenges", *Journal of Supercomputing*, Print ISSN: 0920-8542, Electronic ISSN: 1573-0484, 23 May 2025, Vol. 81, No. 8, Article No. 877, Published by Springer Nature, DOI: 10.1007/s11227-025-07371-y, Available: https://link.springer.com/article/10.1007/s11227-025-07371-y.

[10] Charan Bandi, Soheil Salehi, Rakibul Hassan, Sai Manoj P D, Houman Homayoun *et al.*, "Ontology-Driven framework for trend analysis of vulnerabilities and impacts in IoT hardware", in *Proceedings of the 15th IEEE International Conference on Semantic Computing (ICSC)*, 27-29 March 2021, Laguna Hills, United States, Print on Demand (PoD) ISSN: 2325-6516, pp. 211-214, Published by IEEE, DOI: 10.1109/ICSC50631.2021.00045, Available: https://ieeexplore.ieee.org/document/9364426.

[11] Xiang Chen, Changlin Yang, Yuhong Nan and Zibin Zheng, "An empirical study of high-risk vulnerabilities in IoT systems", *IEEE Internet of Things Journal*, Electronic ISSN: 2327-4662, 27 November 2024, Vol. 12, No. 2, pp. 1590-1601, Published by IEEE, DOI: 10.1109/jiot.2024.3506976, Available: https://ieeexplore.ieee.org/document/10769424.

[12] Ahmad Alomari and Sathish A.P. Kumar, "Securing IoT systems in a post-quantum environment: Vulnerabilities, attacks, and possible solutions", *Internet of Things*, Print ISSN: 2543-1536, Online ISSN: 2542-6605, 26 February 2024, Vol. 25, p. 101132, Published by Elsevier, DOI: 10.1016/j.iot.2024.101132, Available: https://www.sciencedirect.com/science/article/abs/pii/S254266052400074X.

[13] Abdul Razaque, Gulnara Bektemyssova, Meenhoon Khan, Joon Yoo, Meer Jaro Khan *et al.*, "Comprehensive analysis of Li-Fi technology: positioning algorithms, security vulnerabilities, and future IoT applications", *Telecommunication Systems*, Print ISSN:1018-4864, Electronic ISSN: 1572-9451, 24 October 2025, Vol. 88, No. 4, pp. 1-57, Published by Springer Nature, DOI: 10.1007/s11235-025-01358-z, Available: https://link.springer.com/article/10.1007/s11235-025-01358-z.

[14] Pascal Oser, Rens W. van der Heijden, Stefan Lüders and Frank Kargl, "Risk prediction of IoT devices based on vulnerability analysis", *ACM Transactions on Privacy and Security*, Print ISSN: 2471-2566, Electronic ISSN: 2471-2574, 4 May 2022, Vol. 25, No. 2, pp. 1-36, Published by Association for Computing Machinery, DOI: 10.1145/3510360, Available: https://dl.acm.org/doi/10.1145/3510360.

[15] Mounika Baddula, Biplob Ray, Mahmoud ElKhodr, Adnan Anwar and Pushpika Hettiarachchi, "Vulnerability assessment and risk modeling of iot smart home devices", in *Proceedings of the 38th International Conference on Advanced Information Networking and Applications (AINA)*, 9 April 2024, Kitakyushu, Japan, Print ISBN: 978-3-031-57930-1, Online ISBN: 978-3-031-57931-8, pp. 459-469, Published by Springer, DOI: 10.1007/978-3-031-57931-8_44, Available: https://link.springer.com/chapter/10.1007/978-3-031-57931-8_44.

[16] Jian Gao, Xin Yang, Yu Jiang, Houbing Song, Kim-Kwang Raymond Choo *et al.*, "Semantic learning based cross-platform binary vulnerability search for IoT devices", *IEEE Transactions on Industrial Informatics*, 16 October 2019, Print ISSN: 1551-3203, electronic ISSN: 1941-0050, Vol. 17, No. 2, pp. 971-979, Published by IEEE, DOI: 10.1109/tii.2019.2947432, Available: https://ieeexplore.ieee.org/document/8869943.

[17] Yungtai Cheng and Shinming Cheng, "Firmulti Fuzzer: discovering multi-process vulnerabilities in iot devices with full system emulation and VMI", in *Proceedings of the 5th Joint Workshop on CPS and IoT Security and Privacy (CPSIoTSec)*, 26 November 2023, New York, United States, ISBN: 979-8-4007-0254-9, No. 9, pp. 1-19, Published by Association for Computing Machinery, DOI: 10.1145/3605758.3623493, Available: https://dl.acm.org/doi/10.1145/3605758.3623493.

[18] Haoyu Xiao, Yuan Zhang, Minghang Shen, Chaoyang Lin, Can Zhang *et al.*, "Accurate and efficient recurring vulnerability detection for iot firmware", in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 14-18 October, 2024, Salt Lake City, United States, ISBN: 979-8-4007-0636-3, pp. 3317-3331, Published by Association for Computing Machinery, DOI: 10.1145/3658644.367027, Available: https://dl.acm.org/doi/10.1145/3658644.3670275.

[19] Pallavi Sivakumaran, Chaoshun Zuo, Zhiqiang Lin and Jorge Blasco, "Uncovering vulnerabilities of Bluetooth low energy IoT from companion mobile apps with ble-guuide", in *Proceedings of the 18th ACM ASIA Conference on Computer and Communications Security (ASIA CCS)*, ISBN: 979-8-4007-0098-9, 10 July 2023, pp. 1004-1015, Published by Association for Computing Machinery, DOI: 10.1145/3579856.35958, Available: https://dl.acm.org/doi/10.1145/3579856.3595806.

[20] Murat Koca and Isa Avci, "A novel hybrid model detection of security vulnerabilities in industrial control systems and IoT using GCN plus LSTM", *IEEE Access*, Electronic ISSN: 2169-3536, 23 September 2024, Vol. 12, pp. 143343-143351, Published by IEEE, DOI: 10.1109/ACCESS.2024.3466391, Available: https://ieeexplore.ieee.org/document/10689427.