Journal of
Cyber Security

Tech Science Press

https://doi.org/10.32604/jcs.2026.078232

ARTICLE

A Novel Malware Detection Method Based on IPSO-Optimized LSTM

Zheng Yang[1], Hua Zhu[1,*], Zhao Li[2], Gang Wang[3] and Meng Su[1]

[1]Electric Power Research Institute of Yunnan Power Grid Company Ltd., Kunming, China
[2]Department of System Operation, Yunnan Power Grid Co., Ltd., Kunming, China
[3]School of Electrical Engineering, Chongqing University, Chongqing, China

*Corresponding Author: Hua Zhu. Email: zhuhuakm@163.com

Received: 26 December 2025; Accepted: 03 April 2026; Published: 18 May 2026

ABSTRACT: The rapid integration of IoT technologies in modern power systems, while enhancing operational efficiency, has introduced critical cybersecurity vulnerabilities. The proliferation of interconnected terminal devices across diverse operational domains has escalated cybersecurity risks, particularly from sophisticated malware attacks targeting critical grid infrastructure. These threats manifest through Application Programming Interface (API) call hijacking, command injection in industrial control protocols, and evasion of conventional signature-based detection systems. To address these challenges, this paper proposes a novel malware detection framework specifically designed for power IoT ecosystems. First, a malware detection model based on long short-term memory network (LSTM) is established; then, the API call sequence data is preprocessed and mined for malware feature coefficients; next, the improved particle swarm optimized (IPSO) is introduced to optimize the four hyper-parameters of the LSTM; and finally, a publicly available dataset is used in the experimental part to evaluate and validate the proposed IPSO-LSTM malware detection model is evaluated and validated by comparing it with other detection methods, and the results indicate that the proposed method shows excellent detection performance in the experiments and can realize efficient and accurate detection of malware, which provides a strong guarantee for the safe operation of power mobile terminals.

KEYWORDS: Application programming interface; malware detection; IPSO; LSTM; power IoT

1 Introduction

While the Internet of Things (IoT) has grown rapidly in recent years, there has also been a significant increase in malicious attacks targeting IoT infrastructure, applications, and end devices. Malware, the most significant threat in IoT, often leads to problems such as leakage of private personal data and botnet attacks on IoT devices. In particular, for electric power enterprises, with a large number of access to user-side smart appliances with high-power terminals such as air conditioners, heat pumps, and electric vehicles [1–3], the user IoT device terminals establish a closer connection with the electric power system, which, however, also leads to an increase in the cybersecurity threats faced by the electric power system. Therefore, it is of great significance and value to propose efficient and accurate malware detection methods and improve the scientific and reasonable malware detection mechanism to safeguard the security of electric power IoT. However, due to the specificity of electric power enterprises, the electric power mobile terminal application scenarios are quite different from the application scenarios of other mobile terminals, which makes the mobile business software on electric power mobile terminals have different behavioral patterns from the mobile Internet software. Therefore, targeted malware detection methods need to be researched according

to the application scenarios of electric power mobile terminals to increase detection accuracy and improve the network security defense level of electric power IoT terminals.

In the early 21st century, an employee fired from a Texas electric utility company managed to hack into the company's network using his undisabled login, paralyzing the power forecasting system [4]. In 2015, a cyberattack on the Ukrainian power grid, in which attackers gained access to a console to shut down circuit breakers, led to the shutdown of 30 substations, affecting 230,000 people [5]. In 2017, a cyberattack on the UK's power infrastructure, in which miscreants infiltrated management systems in order to shut down parts of the grid [6]. Such attacks threaten smart grid reliability, and attackers may also disrupt communications, tamper with data, or flood networks, limiting operators' ability to monitor. In the context of the construction of the Internet of Things (IoT) of electric power, with the rapid development of the smart grid, the operation and management of the electric power system is experiencing unprecedented changes. As its core component, the Internet of Things (IoT) of electric power realizes the intelligence, informatization, and networking of all aspects of electric power production, transmission, distribution, and consumption by integrating advanced sensing technology, communication technology, data processing technology, and intelligent control technology [7]. However, this change also brings unprecedented security challenges. As power enterprise net-works gradually become open, attackers can launch penetration attacks on terminals through the public network, implant malware to commit malicious behaviors and cause a series of security threats [8].

The challenge of network openness: with the popularization of power IoT, the network structure of the power system is becoming increasingly open, and more and more devices are interconnected through the public network. While this openness increases the flexibility and efficiency of the system, it also provides opportunities for attackers to exploit [9]. By simply using the Internet, attackers have the potential to find weaknesses in the power system and launch penetration attacks against end devices. Diversity of terminal devices: the power IoT contains many intelligent terminal devices, such as smart meters, sensors, controllers, etc. [10]. The diversity of these devices in terms of hardware, operating systems, communication protocols, etc., makes unified security management extremely difficult. Once a device has a security vulnerability, it may become an entry point for attackers to invade, affecting the security of the whole power system [11]. Threat of malware: Once an attacker succeeds in infiltrating the power IoT system, he or she may implant a variety of malware, such as viruses, Trojans, ransomware, and so on. This malware can perform various malicious behaviors, including but not limited to data theft, tampering, disrupting the normal operation of the system, and even causing damage to physical equipment, which brings serious economic losses and social impacts to the power system [12–17].

Depending on the features required to train the detection model, the existing methods for malware detection can be categorized into static feature-based detection and dynamic feature-based detection [18]. Static analysis is a code analysis technique that takes as input the source or binary code of a program. It examines this code without running the program to verify the safety, reliability, etc., of the program [19]. In static analysis, static feature extraction is a method of extracting features from content without running an executable file. The core advantage of static analysis over dynamic analysis is that it does not require the running of a program, which makes the process more efficient and faster. Given these characteristics, static analysis techniques are widely used in identifying malware. For example, Sihag et al. [20] introduced a novel malware detection and family classification system called BLADE (roBust maLwAre DEtection system), which is built based on Operational Segment Documents (OSDs) and is efficient, accurate and resistant to obfuscation. BLADE is evaluated to perform well on several benchmark datasets and effectively detects samples from multiple obfuscation techniques. Şahin et al. [21] applied two dimensionality reduction techniques, PCA and LDA, in Android malware detection to make it effective for real-time malware

detection. In addition, the authors of [22] proposed a new approach for Android malware obfuscated variant detection-MGOPDroid, which employs novel feature selection algorithms and classification models to achieve malware detection based on multi-granularity opcode features. However, due to the dynamic nature of threats in the online world, traditional machine learning-based malware detection solutions have proven inadequate. New types of malwares, relying on static feature extraction, often evade detection by most machine learning-based security solutions [23].

Dynamic analysis is a technique that monitors application behavior at runtime, capturing the actual impact of code execution such as user log changes, registry updates, system file additions and deletions. Through this process, behavioral features are extracted and transformed into features that can be used by deep learning models [24]. However, since dynamic analysis only focuses on the behavioral information of the actual execution, it cannot fully capture the overall characteristics of the code. Casolare et al. [25] proposed to represent an application with images obtained from system call traces. By executing the analyzed application in a virtual environment, system call traces are extracted and images of the application execution are generated. Sihag et al. [26] proposed a deep learning-based malware detection scheme that uses dynamic analysis to capture logs that characterize the behavior of running applications. The captured logs specifically describe system call tracing, binders analysis, network traffic capture, and composite behavioral interactions. The advantage of dynamic analysis is that it is independent of code obfuscation and encryption and can be based on the malicious behavior of an application. However, the disadvantages are that it consumes more system resources than static analysis, requires higher requirements for malware analysts in terms of technical skills, etc., and does not apply to the testing of large-scale applications [27].

In recent years, machine learning and deep learning have shown excellent performance in feature extraction, recognition and classification, and bioinformatics [28,29]. In the field of cybersecurity protection, Ref. [30] employes a variant of the Antlion Optimization Algorithm (ALOA) for rigorous hyperparameter tuning to construct Extreme Gradient Boosting (XGBoost) and B models for identifying and classifying common IoT botnet threats, successfully hacking nine industrial-grade IoT devices. Ref. [31] proposes a malware detection method for intrinsic semantics, in this method, the API semantic features and API co-occurrence features are combined to improve the malware detection performance. In addition, Ref. [32] introduces an Android malware detection algorithm, which is based on a hybrid deep learning model that combines a deep belief network (DBN) with a gate recurrent unit (GRU) to process static and dynamic behavioral features of the malware, respectively, and ultimately outputs the classification results through a neural network. Lian et al. [33] designed a multi-input deep learning model that has the ability to process multi-dimensional digital and image features. It realizes multitasking by learning three sub-models in parallel and an integrated specific sub-model, and supports parallel computing across devices. Xing et al. [34] proposed a malware detection method using an autoencoder in deep learning. The model combines the grayscale image representation of malware with a network of autoencoders in a deep learning model, analyzes the feasibility of the malware grayscale image method based on the reconstruction error of the autoencoders, and realizes the classification of malware and benign software by using the downscaling features of the autoencoders. Combined with the core challenges faced by the safe operation of power mobile terminals, for the shortcomings of existing malware detection methods, this paper adopts the IPSO to optimize LSTM, and proposes a malware detection method based on feature extraction of API call sequences, with the following specific contributions.

In this paper, the particle swarm optimization algorithm is improved by adopting a multilinear decreasing weights strategy, which dynamically adjusts the global search ability and local search ability of particle optimization, so that the algorithm can find the global optimal solution faster and more accurately; In this paper, the proposed improved particle swarm algorithm is introduced into the LSTM network, and the

model hyperparameters of the LSTM are synchronously optimized, which improves the model's detection, recognition, and classification performance; Use the public dataset to train and test the IPSO-LSTM model, and constantly update the model parameters by loss calculation to improve the detection accuracy of the model and realize the efficient detection of malware.

The rest of the paper is organized as follows. The second part describes the architecture and the malware detection process for power IoT. The third part is an intro-duction to the IPSO-LSTM model. The fourth part is the analysis of the arithmetic examples and finally the conclusion and prospect.

2 Malware Detection Process

2.1 Power IoT Architecture

Power IoT architecture comprises three core tiers: application interfaces for intelligent operations, network transmission for data flow, and sensor components for environmental monitoring, as visualized in Fig. 1.

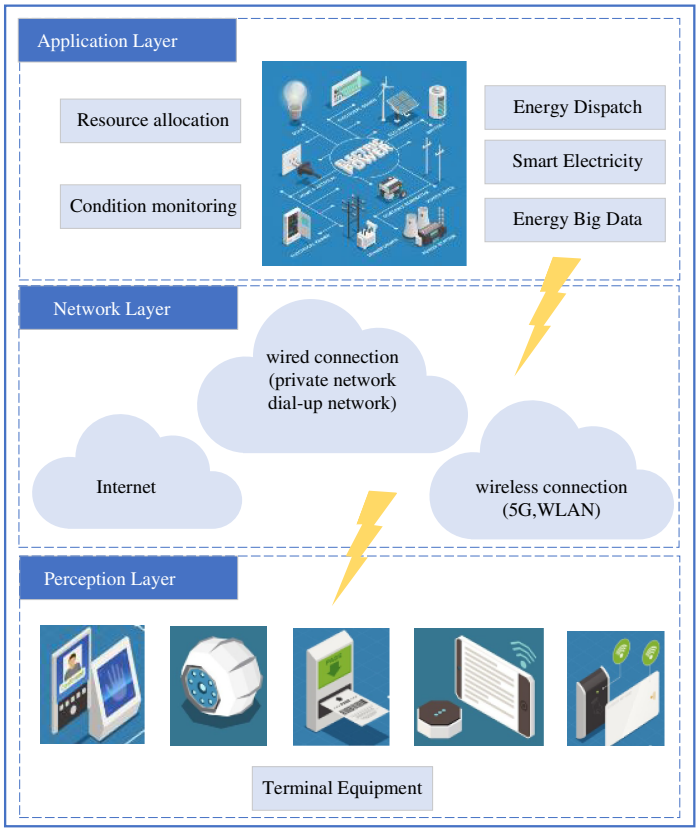


Figure 1: Power IoT architecture.

2.1.1 Application Layer

The application layer is the value realization layer of electric power IoT, which is mainly responsible for transforming the collected data into practical applications to provide intelligent support for grid operation, equipment management, and user services. The application layer can be divided into two parts: internal business and external business. Internal business includes: (1) using IoT technology to optimize resource allocation, reduce operating costs and improve economic efficiency; (2) real-time monitoring of grid status,

preventing failures, and guaranteeing the safe and stable operation of the power grid; (3) optimizing energy scheduling, improving the utilization rate of clean energy, and promoting the development of green energy. External business includes: (1) providing comprehensive energy services, such as smart home, smart electricity, etc., to meet the diversified needs of users; (2) expanding new business areas based on IoT technology, such as energy finance and energy big data.

2.1.2 Network Layer

The network layer functions as the central data conduit in power IoT systems, facilitating bidirectional transmission between sensor-generated datasets and analytical processing modules. The network layer mainly realizes the following functions:

Construct a cooperative and integrated power ubiquitous communication network: Through the integrated network architecture of "air and sky", it realizes the extensive connection and data transmission of power-related devices. The system employs hybrid connectivity solutions encompassing fiber-optic infrastructure and domain-specific wireless grids (e.g., power-dedicated radio networks) to ensure robust data transmission across heterogeneous operational environments.

Deployment of power-dedicated radio networks: Addressing pervasive IoT terminal distribution and geographically dispersed infrastructure, this initiative targets resolving critical last-mile connectivity challenges.

Data transmission and security: The network layer needs to ensure the real-time, reliability and security of data transmission. Through encryption technology, authentication mechanism and other security measures, to protect the information in the process of data transmission from being leaked or tampered with.

2.1.3 Perception Layer

The perception layer serves as the cyber-physical infrastructure of power IoT systems, executing real-time operational metrics acquisition across critical grid components through embedded sensing nodes. The perception layer mainly includes the following aspects:

Data collection devices: including various types of sensors, smart meters, mutual transformers, concentrators and other devices, which are used to collect a variety of data during the operation of the power grid, such as voltage, current, power, temperature, humidity and so on.

Edge intelligence: computational nodes embedded at the perception layer execute localized data preprocessing and feature extraction, minimizing upstream data payload transmission while optimizing processing latency. This helps to reduce the pressure on the cloud computing center and enables real-time response and decision support.

Expanding the range of information reception: with the deepening of the construction of electric power IoT, the construction of the perception layer needs to continuously expand the range of information reception. This includes measures such as adding new data collection equipment, optimizing the layout of equipment, and improving the performance of equipment, in order to achieve comprehensive perception and accurate monitoring of the power grid.

Differing from the traditional Internet architecture, the Internet of Things architecture has the following characteristics: (1) more mobile terminal devices in the perception layer, and the topology of the network changes quickly; (2) more types of devices in the perception layer, and there is a great deal of heterogeneity; (3) more access to the wireless devices; (4) a wider range of applications; (5) higher requirements for data security; and (6) more independence between the various terminals in the network.

The specificity of the structure of the IoT leads to the existence of multiple device environments in the IoT, and poses certain challenges to the detection of malware in the IoT. Malware detection in IoT can be either at the server or at the terminal, and the design of intelligent detection algorithms is the key to detecting malware in IoT.

2.2 API in the Power Grid

With the popularization of cloud computing and big data technologies, business architecture is gradually transforming into a model that relies on platform support, lightweight applications and microservice architecture. In order to accelerate system deployment and promote business data circulation, especially cross-organizational and business domain data exchange, the application scenarios of API interfaces are getting richer and richer, and they have become one of the core components of application development. In the electric power industry, the role of API interfaces is especially critical, providing efficient data communication and functional access for various applications. API interfaces play a crucial role in the data interaction of power grid business, and their specific applications cover energy data management, business collaboration, internal management and other aspects [35]. The following is a detailed description of the specific application of the API interface in power grid business data interaction.

2.2.1 Data Administration

Data collection and transmission: The API interface can connect various hardware devices and software systems in the power grid, such as smart meters, sensors, renewable energy generation facilities, etc., and collect energy data generated by these devices in real time. The collected data is transmitted to the grid company's data center or cloud platform through the API interface for storage, analysis and processing.

Data sharing and access: The API interface allows grid enterprises to share data with other external applications or systems (e.g., smart city applications, smart transportation systems, smart home systems, etc.). Through the API interface, external applications can access the grid enterprise's energy data for energy management, dispatch optimization, environmental monitoring and other purposes.

Data analysis and mining: Grid companies can use API interfaces to integrate heterogeneous energy data from multiple sources, and use big data technology for analysis and mining. Through data analysis, the laws, trends and problems of energy use can be discovered, providing a scientific basis for grid planning, operation and maintenance decisions.

2.2.2 Operational Collaboration

Smart home system docking: Through the API interface, power grid enterprises can realize docking with smart home systems. The smart home system can obtain real-time power data and home appliance operation status data from the grid enterprise, and intelligently control and manage home appliances based on these data to realize energy saving and intelligence.

Intelligent transportation system docking: Grid companies can provide real-time accurate data such as the location of power vehicles and power consumption to the intelligent transportation system through the API interface. The intelligent transportation system can carry out intelligent scheduling and management of power vehicles based on these data to improve operational efficiency and service quality.

Smart environmental protection system docking: Through the API interface, power grid companies can provide power consumption data and power emission data from industrial enterprises to the smart environmental protection system. The smart environmental protection system can use these data to accurately monitor and warn of environmental pollution and develop targeted environmental protection measures.

2.2.3 Internal Management

Grid dispatch automation: In the grid dispatch automation system, the API interface is used to monitor and control the operating status of the grid in real time. Dispatchers can use the API interface to obtain the operating data of core equipment, analyze the trend fluctuations of the grid, and make real-time dispatch decisions based on load changes.

Equipment maintenance and fault warning: Through the API interface, power grid enterprises can realize remote monitoring and maintenance of power grid equipment. When equipment malfunction or abnormality occurs, the API interface can send real-time warning information to the relevant personnel, so as to carry out timely maintenance and processing.

Data analysis and report generation: Grid enterprises can utilize API interfaces to integrate internal data for operational analysis and report generation. These reports can help enterprises understand the operation status, identify problems and formulate improvement measures.

API popularity also brings network security risks, hackers can use API vulnerabilities to carry out attacks, steal data or damage the system. API call sequences, as a meticulous mapping of software functional behaviors, and their characterization can reveal the behavioral characteristics of malware. During the execution of various types of malicious attacks, malware generates rich API call sequences, which constitute the key data basis for malware detection. Through in-depth analysis of these API call sequences, API call sequences with malicious attack characteristics can be identified by combining their behavioral characteristics, and further based on the comparison of their behavioral characteristics with historical data, the detection and identification of different malware can be realized [36], thus ensuring the safe and stable operation of multiple intelligent terminals within the power IoT.

2.3 Deep Learning-Based Malware Detection Process

Deep learning-driven malware detection is essentially learning the features of the data, learning by fitting the feature space, feeding the data into the fitted model for classification or clustering, and ultimately achieving the effect of detection. In this paper, the detection of malware is defined as five parts, and the specific process is shown in Fig. 2.

Combined with the detection process shown in Fig. 2, the common steps for malware detection in IoT include:

(1) Data collection. It is used for the collection of malware samples in IoT, which is collected and used for the training and testing of the detection model, and the data collection is related to the performance of the detection model. Comprehensive considerations should be taken into account while performing data collection, such as whether the data is valid or not, and whether the diversity of data samples is sufficient or not. The detection of malware in this paper requires the collection of both malware samples as well as malicious traffic generated by malware.

(2) Data preprocessing. Used to process the collected dataset to make the data more compatible with the input and training requirements of the model. For malware samples in IoT, the preprocessing section needs to focus on solving problems such as irregular sample names, duplicate samples, and uneven distribution of sample types. For network traffic data in IoT, the focus in the preprocessing part is on solving problems such as message duplication, incomplete messages, network flow aggregation, and message statistics.

(3) Constructing detection models. It is used to construct machine learning, deep learning models for the detection of malware. Model construction is the core part of the whole system framework, the

constructed classifier is used to classify the input samples, and the classifier is necessary to achieve malware detection.

(4) Model training, evaluation, optimization and improvement. This step is the key part of the whole system architecture and will be preprocessed data as input to the constructed model for model training, in the training process the parameters of the model are continuously adjusted and optimized until the end of model convergence (specific case analyses are discussed in Section 4).

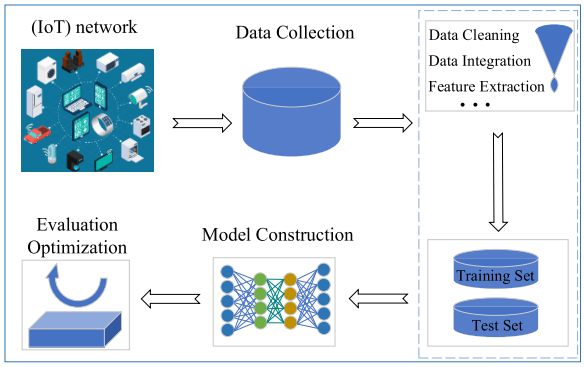


Figure 2: Malware detection flowchart.

3 IPSO-LSTM Model

3.1 LSTM

LSTM denotes a recurrent neural network with long/short-term memory architecture, operationally analogous to Recurrent Neural Network (RNN). As shown in Fig. 3, A represents the basic structural unit of recurrent neural network-recurrent kernel, and every time a new input x is read into the recurrent kernel block, the state h will be updated. LSTM incorporates gated mechanisms to regulate temporal dependencies within RNN architectures; this structural enhancement mitigates long-sequence gradient vanishing while enhancing sequence modeling efficacy. Fig. 4 demonstrates the LSTM's gated recurrent architecture, comprising three core modules: forget gate, input gate, and output gate.

In Fig. 4, σ represents the activation of the sigmoid function and tanh represents the activation of the tanh function. The forget gate activates the sigmoid function by splicing the cyclic kernel state h_{t-1} at the moment t − 1 and the input data x_t at the moment t. It generates the data between 0–1, which represents the ratio of the output C_{t-1} at the moment t − 1 retained at the moment t. The sigmoid function is activated at the moment t − 1 by splicing the input data x_t.

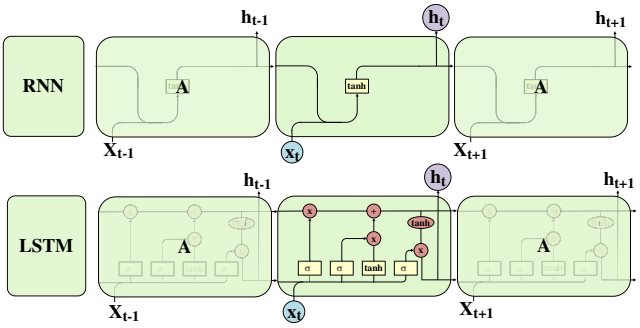


Figure 3: Long short-term memory network.

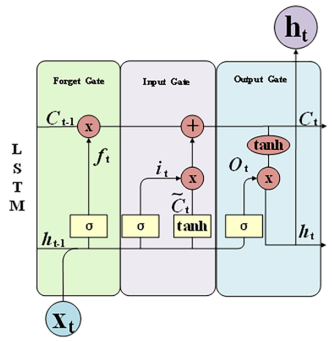


Figure 4: The gate control unit structure of LSTM.

The input gate is divided into two parts, the sigmoid function determines the ratio of data update; the tanh function generates the candidate value $\tilde{C}t$ at the moment t. The output gate generates a new high-dimensional vector from the cyclic kernel state h_{t-1} at the moment t and the input data x_t at the moment t, and then passes through the sigmoid function to get the corresponding output ratio; the data on the conveyor belt passes through the tanh function to get the between −1 and 1 vector; finally, the loop kernel state h_t at moment t is determined. Where, the three gates of the LSTM are calculated as follows:

$$f_t = \sigma\left(w_f \cdot h_{t-1} + w_f \cdot x_t + b_f\right) \tag{1}$$

$$i_t = \sigma\left(w_i \cdot h_{t-1} + w_i \cdot x_t + b_i\right) \tag{2}$$

$$o_t = \sigma\left(w_o \cdot h_{t-1} + w_o \cdot x_t + b_o\right) \tag{3}$$

where, f_t, i_t, o_t are the corresponding gate state operation results, all are numbers between 0 and 1, representing the corresponding output ratios; w_f, w_i, w_o are the corresponding gate weight matrices, which are based on the correlation coefficient matrices generated during the training of the model; b_f, b_i, b_o are the corresponding gate bias terms, which belong to the trainable parameters of the model, as do the gate weight matrices. The outputs of the LSTM cyclic kernel layer at the moment t are determined by the state of the conveyor belt and the output gate are determined by the following formula:

$$\tilde{C}_t = \tanh\left(w_C \cdot h_{t-1} + w_C \cdot x_t + b_C\right) \tag{4}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{5}$$

$$h_t = o_t \cdot \tanh\left(C_t\right) \tag{6}$$

where, $\tilde{C}_t$ is the state of the cyclic core unit at the moment t; w_C is the state weight matrix of the cyclic core layer; and b_C is the state bias term of the cyclic core layer.

3.2 IPSO-LSTM

3.2.1 IPSO Algorithm

Particle Swarm Optimization (PSO) facilitates model parameter optimization in malware detection research. In the PSO algorithm, the optimization direction of the particle (potential solution) in the D-dimensional goal search space (solution space) is affected by three factors: the speed of the last iteration, individual cognition and group orientation. Individual cognition is that the particle adjusts the optimization direction of the next iteration according to its own learning; group orientation is that the particle adjusts the optimization direction of the next iteration according to the optimal course of action of the group. As

a result, parameters such as inertia weight and acceleration factor can be introduced to describe the speed update and position update of the *i*th particle:

$$V_{ij}(k+1) = \omega V_{ij}(k) + c_1 r_1(k)\left[p_{ij}(k) - x_{ij}(k)\right] + c_2 r_2(k)\left[p_{gj}(k) - x_{ij}(k)\right] j \in \mathrm{D} \tag{7}$$

$$x_{ij}(k+1) = x_{ij}(k) + V_{ij}(k+1)\, j \in \mathrm{D} \tag{8}$$

where, $V_{ij}(k)$ represents the *j*-dimensional velocity component of the *i*th particle at the *k*th iteration; ω is the inertia weight of the particle iteration; c_1, c_2 are the acceleration factors of the particle iteration; r_1, r_2 represent random numbers between 0 and 1; $p_{ij}(k)$ represents the *j*-dimensional component of the local optimal position of the *i*th particle at the *k*th iteration; $x_{ij}(k)$ represents the *j*-dimensional position component of the *i*th particle at the *k*th iteration; $p_{gj}(k)$ represents the *j*-dimensional component of the overall optimal position of the *i*th particle at the *k*th iteration.

In the traditional PSO algorithm, the particles in the solution space update their flight speed and position according to their own search trajectories and those of other particles. In this paper, an improved particle swarm algorithm is proposed by optimizing the inertia weights and acceleration factors. These parameters are dynamically adjusted and can be optimized with the increase in the number of iterations, with the specific formula as follows:

$$\omega(k) = \omega_{\max} - (\omega_{\max} - \omega_{\min})(k/T_{\max})^2 \tag{9}$$

$$c_1(k) = c_{10} - (c_{10} - c_{11})(k/T_{\max}) \tag{10}$$

$$c_2(k) = c_{20} - (c_{20} - c_{22})(k/T_{\max}) \tag{11}$$

where, $\omega(k)$, $c_1(k)$ and $c_2(k)$ represent the inertia weights and acceleration factor of the particle at the *k*th iteration, $\omega_{\max}$ and $\omega_{\min}$ represent the maximum and minimum values of the inertia weights, and c_{10}, c_{20} and c_{11}, c_{22} represent the initial and final values of the acceleration factor, respectively. Specifically, the design of multi-linear decreasing inertia weights enables PSO to initiate global exploration of the hyperparameter space from a higher value. Nonlinear descent allows for a relatively extended exploration phase before transitioning rapidly to a lower for subsequent iterations focused on local refinement, thereby helping to avoid premature convergence. The adaptive acceleration coefficient ensures that, as the search progresses, particles gradually shift their dependence from their own historical best positions to the global optimum within the swarm. This promotes initial diversity and ultimately fosters consensus. By incorporating these enhanced strategies, the PSO algorithm can more rapidly identify optimal solutions for model hyperparameters, thereby improving computational efficiency and detection performance.

3.2.2 IPSO-LSTM Model

The proposed IPSO-LSTM architecture for malware detection employs normalized API call sequences as feature vectors, with cross-validation-driven stratified partitioning of malware behavioral data into training-test subsets. Before training the model, IPSO is introduced to optimize four hyper-parameters of the LSTM model: hidden layer size (units), learning_rate, batch_size, dropout_rate, and to trade-off between the particle's ability to search for optimal results and local search. The ability of global search and local search addresses inherent constraints in conventional optimization optimizers to a certain extent. Adjusting hyperparameters operates by effectively enhancing the critical diagnostic model's diagnostic efficacy, such as the optimization of batch size, which can make the direction of model gradient descent more accurate, thus improving the training speed. The operational pipeline of the cyber threat intelligence architecture integrating IPSO-LSTM is shown in Fig. 5, and the specific diagnostic validation as follows:

(1) Read the API call sequence data, perform data preprocessing, and divide it into test set and training set according to the ratio of 8:2 (It should be noted that this test set is not utilized at all during the entire model development and hyperparameter optimization process; it is reserved solely for the final performance reporting);
(2) Initialize the hyperparameter optimization interval and step size, configure the swarm cardinality parameters and experimental iterations for PSO optimization;
(3) Input the training set, LSTM is trained according to the data in the training set to construct the malware detection model;
(4) Randomly generate the hyperparameter particle population in the optimization interval according to the step size and input the results into the LSTM model;
(5) Search for the optimal particles of the population and calculate the detection and recognition accuracy, i.e., the particle fitness;
(6) Compare the current particle fitness with the global optimal particle fitness, update and record the global optimal particle;
(7) Update the particle iteration speed, position and inertia weights according to Eqs. (7)–(9);
(8) Check if the program has run the set number of times. If not, repeat steps (4) (5) (6) (7); if yes, input the test set, test the model performance, and output the optimization results and diagnosis accuracy.

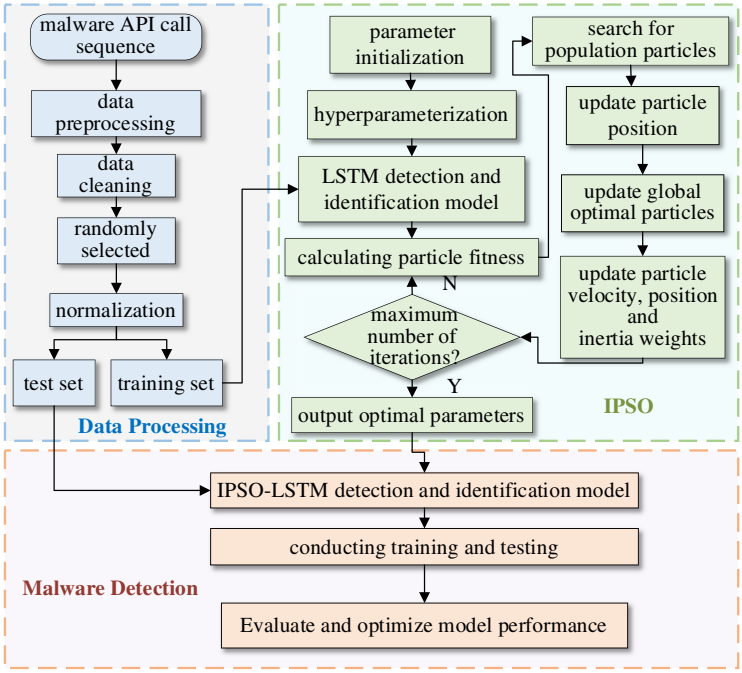


Figure 5: IPSO-LSTM detection and identification process.

Additionally, to more clearly describe the parameter configuration and algorithm execution flow of the proposed IPSO-LSTM algorithm during malware detection, we provide a short pseudocode for IPSO-LSTM in Algorithm 1.

Algorithm 1: IPSO-LSTM for malware detection

Input:

1) Dataset D = {(X_i, y_i)} where X_i is API call sequence, y_i ∈ {1, . . ., 8} is malware family label.
2) Hyperparameter search spaces (see Table 1).
3) IPSO parameters: population_size = 20; max_iter = 30; ω_{max} = 0.9, ω_{min} = 0.4;
c_{10} = 2.5; c_{20} = 0.5; c_{11} = 0.5; c_{22} = 2.5; λ = 0.001
4) Fixed random seed = 42 for all stochastic operations.

Output:

1) Trained IPSO-LSTM model with optimal hyperparameters.
2) Performance metrics for Malware Detection.

Abbreviated Procedure:

Data Preparation:
1) Set random seed = 42.
1: 2) Stratified 5-fold split: Partition D into 5 disjoint folds (G1, . . ., G5) preserving class distribution.
3) For each fold as test set, combine remaining 4 folds as training data → 5 evaluation runs.
IPSO Hyperparameter Optimization (for a given training set):
1) Initialize population P of size 20 with random particles.
Each particle position p = (h_hidden, h_lr, h_batch, h_dropout) within bounds.
Each particle velocity v initialized randomly in [−step, step] per dimension (step = (ub − lb)/10).
2) For iteration k = 1 to max_iter:
Build LSTM model with hyperparameters p_i
Perform 5-fold cross-validation on training set
2: Record training time t_i for the CV process.
Compute fitness fit_i = F1_cv − λ * t_i (Eq. (16)).
Update personal best pbest_i if fit_i > pbest_fitness_i
Update global best gbest if fit_i > gbest_fitness.
Update inertia weight ω(k) = ω_max − (ω_max − ω_min)*(k/max_iter)^2 (Eq. (9)).
Update acceleration coefficients: (Eq. (10))
c1(k) = c10 − (c10 − c11)*(k/max_iter)
c2(k) = c20 − (c20 − c22)*(k/max_iter)
3) Return gbest (optimal hyperparameters).
Final Model Training and Evaluation (for each fold):
1) Let train_data = 4 folds, test_data = 1 held-out fold.
2) On train_data, perform IPSO optimization (Step 2) to obtain best hyperparameters H* (hidden layer size, learning_rate, batch_size, dropout_rate).
3: 3) Retrain LSTM on full train_data using H* with same optimizer settings.
4) Evaluate final model on test_data:
Compute precision, recall, F1 per class, macro-F1, weighted-F1, and confusion matrix.
5) Repeat for all 5 folds.
6) Report mean and standard deviation of metrics across 5 folds.
4: Output Results.

Table 1: Super parameter optimization interval.

Hyperparameter	Euclidean Interval	Search Steps
Hidden layer size	[40, 80]	[−4, 4]
Learning rate	(0.1, 0.01, 0.001)	1
Batch_size	[8, 30]	[−2, 2]
Dropout rate	[0.2, 0.5]	[−0.01, 0.01]

4 Analysis of Examples

4.1 Experimental Design

The experimental setup in this study leverages Python and the TensorFlow framework for deploying the introduced malware detection system. The comparison is performed on a data workstation equipped with a CPU model Inter(R) Core (TM) i7-10700, 2.90 GHz, 16 GB of RAM, and an NVIDIA GTX2080 GPU with 4G video memory. The comparative benchmark models involved in the experiments are Convolutional Neural Network (CNN) [37], Gate Recurrent Unit (GRU) [38], and XGBoost [39]. The experiments in the paper use the public dataset Mal-API-2019 [40] by Turkish scholars, a public malware dataset generated by Cuckoo Sandbox based on the analysis of Windows system API calls, and is available for deep learning algorithms to be trained for malware identification. The dataset records the API call logs of eight types of mal-wares. These are analyzed to mask their illegal behavior by analyzing the meaningless opcodes of malicious soft additions such as artifacts and assembly parts. Besides, the key implementation parameters and settings for the IPSO-LSTM model proposed in this study are as follows:

(1) API Call Sequence Encoding: All unique API function names in the dataset are constructed into a vocabulary. Each API call is mapped to an integer index within this vocabulary, and each index is converted into a 128-dimensional dense vector.
(2) Sequence Length and Padding: After analyzing all API call sequences in the dataset, the maximum sequence length is set to 500. Sequences shorter than 500 are padded with zeros at the end; sequences longer than 500 are truncated, retaining the last 500 calls, as recent calls are typically more representative of the current behavioral state.
(3) Optimizer and Loss: The Adam optimizer was selected as the base optimizer for the LSTM model. Its initial learning rate (learning_rate) is one of the hyperparameters optimized by the IPSO algorithm (search range shown in Table 1). Model training employs a categorical cross-entropy loss function.
(4) Randomness Control: We uniformly set the random seed for TensorFlow to 42.
(5) The model is trained as an 8-class classifier, corresponding to the 8 malware families in the Mal-API-2019 dataset.

The dataset studied in this paper focuses on malware threats for Microsoft systems, covering API call information including eight types of malwares such as Spyware, Trojan, and Worms. Fig. 6 shows the distribution of the data samples of the eight categories of mal-ware included in Mal-API-2019, totaling 7107 samples, of which, the Adware mal-ware has the lowest number, this is since the real adware revenue depends on the number of user views and downloads, and the implantation of malware will affect the software's word of mouth, resulting in the decrease of user views and downloads, so in the advertisement series of software there are not exist too much malware. The samples of the remaining seven categories of malware are nearly evenly distributed.

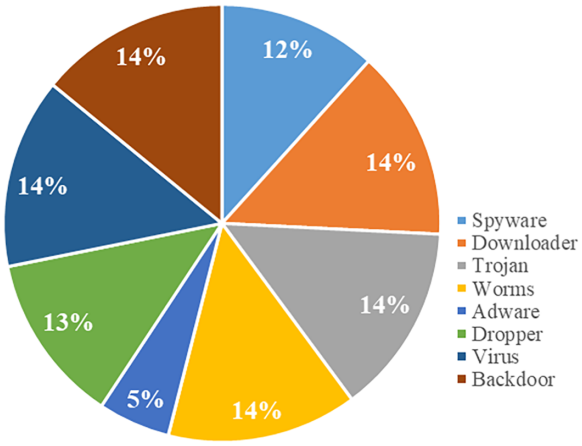


Figure 6: Mal-API-2019 dataset distribution map.

4.2 Data Preprocessing

Data preprocessing operations must also be performed on the data samples before training the model. First, data masking should be applied to the data samples. Specifically, before the malware API calls the feature input model, it is necessary to mask some of the data, input the unmasked part of the data into the model for training, and then predict the masked data through the model after the training is completed, to achieve the purpose of verifying the model's malware recognition results.

The purpose of data masking treatment is similar to that of the attention mechanism, both of which help the model to improve the feature grasping ability of the mod-el. In the paper, the Mal-API-2019 dataset is masked with 20%, and among the 1421 sets of masked samples, 80% of the data are set to −1, 10% of the data are set to random numbers, and the remaining 10% of the data are not processed in any way. This allows the model to learn rich, contextual representations of API call sequences before fine-tuning on the labeled classification task, it helps the model understand the semantic relationships between API calls, leading to more robust feature extraction for classification.

Finally, to better compare the malware recognition performance of different models, the paper comprehensively evaluates the performance of the models in terms of *Precision*, *Recall*, and F_1 scores, as well as the confusion matrices of the eight types of malware recognition results. The specific formula is:

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{14}$$

In the multi-classification task addressed in this paper, F_1 scores can be further categorized into Macro-F_1 and Weighted-F_1, denoted as F_1^M and F_1^W, respectively. The corresponding calculation formulas are as follows:

$$F_1^M = \frac{1}{m}\sum_{i=1}^{m} F_1(i) \tag{15}$$

$$F_1^W = \sum_{i=1}^{m} \frac{N_i}{N} * F_1(i) \tag{16}$$

where, m denotes the number of categories, with $m = 8$ in this paper; $F_1(i)$ represents the F_1 scores for the ith category; N denotes the total number of samples; N_i denotes the number of samples in the ith category.

4.3 Calculus Analysis

In a multi-objective classification task, the classification performance of a model not only depends on its network architecture but is also determined by its hyperparameters. For this reason, the present work implements a metaheuristic optimization framework for systematically refining the model's hyperparameter configuration, thereby elevating computational detection efficacy against polymorphic malware variants. Leveraging swarm intelligence principles, the PSO-based optimization framework navigates the solution landscape through dynamic modulation of swarm individuals' positional dynamics and convergence thresholds. This metaheuristic process enables automated hyperparameter configuration for adversarial malware detection systems while ensuring Pareto-optimal performance realization in polymorphic threat scenarios.

Nevertheless, canonical swarm optimization methodologies exhibit inherent limitations in escaping suboptimal attractors due to premature convergence characteristics, particularly when navigating high-dimensional non-convex solution landscapes prevalent in adversarial malware detection scenarios. This study enhances PSO via dynamic inertia-acceleration adaptation, with optimized LSTM hyperparameter ranges specified in Table 1 derived from adversarial gradient constraints. The optimality-seeking dimension is 4 (the position vector of a particle in PSO has a dimension of 4, corresponding to the four parameters to be optimized in Table 1).

Reference [41] employs a self-supervised training algorithm to estimate the next representation of data points, then constructs a loss function using these points to enhance the training of the proposed CNN classification algorithm. Similarly, we refine the fitness function of the PSO to improve the training effectiveness of the LSTM model. The LSTM is trained using the Mal-API-2019 dataset, and the fitness of the IPSO algorithm is proposed by combining the F_1 scores of the malware recognition results and the training time, with the fitness metric derivation mathematically formalized in Eq. (16) under conjugate duality constraints.

$$fit = F_1 - \lambda \cdot t \tag{17}$$

where *fit* is the population fitness; λ is the penalty factor, and t is the training time. Particularly, in the IPSO hyperparameter optimization loop, we employ K-fold cross-validation (K = 5) on the training set. Specifically, for each set of candidate hyperparameters (particles), we perform 5-fold cross-validation on the training set for the corresponding LSTM model and compute the average validation F_1 score as its fitness value.

4.3.1 Comparison of Hyperparameter Optimization Results

In the experimental design of this section, the comparison models we set up include: standard LSTM (default hyperparameters), random search-optimized LSTM, PSO-LSTM, and IPSO-LSTM (this paper). Additionally, all models utilize the dataset introduced in Section 4.1. Specifically, we divided the original 7107 data samples into five groups for experimentation, ensuring that the proportion of different sample types within each group perfectly matched the distribution shown in Fig. 6. Based on this, we conducted experiments comparing hyperparameter optimization results across the four models designed earlier. It is worth noting that the five data groups were tested on each model individually, meaning each model underwent five tests. The final test results represent the average of these five tests and are statistically presented as (mean ± std). The experimental results are shown in Table 2.

Table 2: Experimental results display (5 groups-mean ± std).

Models	*Precision*	Macro-F_1	Weighted-F_1
Standard LSTM	0.78 ± 0.015	0.75 ± 0.02	0.77 ± 0.01
Random Search-LSTM	0.86 ± 0.008	0.83 ± 0.01	0.84 ± 0.01
PSO-LSTM	0.88 ± 0.006	0.86 ± 0.01	0.87 ± 0.01
IPSO-LSTM	0.91 ± 0.004	0.90 ± 0.01	0.91 ± 0.01

As shown in Table 2, the performance of the LSTM model demonstrates a progressive and significant improvement as the hyperparameters are upgraded from the "default" setting to the "optimization strategy based on IPSO". Specifically, the standard LSTM (default) achieved a precision of only 78.2 ± 1.5%, macro F_1 of 0.75 ± 0.02, and weighted F_1 of 0.77 ± 0.01. This indicates that without hyperparameter optimization, the model's generalization capability is severely constrained. Random Search-LSTM, through random sampling of 50 parameter sets, improved precision to 85.6 ± 0.8%, with macro F_1 and weighted F_1 rising to 0.83 ± 0.01 and 0.84 ± 0.01, respectively; After introducing PSO for hyperparameter optimization, the LSTM model's performance was further enhanced, with all relevant evaluation metrics increasing.

4.3.2 Ablation Experiment

In the ablation experiment design of this section, we constructed a series of variant models using the variable control method to quantitatively analyze the impact of three factors (IPSO, masking operation, time penalty term) on model performance. The model variants are shown in Table 3. Similarly, all models in this section employ the same dataset and testing methodology as described in Section 4.3.1. We present the comparative results of the ablation experiments using bar charts, as shown in Fig. 7.

Table 3: Model variant configuration for ablation experiments.

Model Variants	Configuration Instructions
IPAO-LSTM	The method proposed in this paper: IPSO optimization of hyperparameters + masked pre-training task + fitness function incorporating a temporal penalty term.
Variant 1 (V1)	Replace IPSO with PSO for hyperparameter optimization, with all other settings identical to the full model (masking + temporal penalty).
Variant 2 (V2)	Remove masking operations; otherwise, identical to the full model (IPSO + time penalty).
Variant 3 (V3)	The fitness function removes the time penalty term, while other components remain identical to the complete model (IPSO + masking).

In Fig. 7, "Iterations" refers to the number of iterations at convergence, includes the sum of the iteration counts for five sets of data. We compared the performance of models V1, V2, and V3 against the detection capabilities of IPSO-LSTM. It is evident that: (1) Compared to "V1", the IPSO-LSTM model achieved higher precision, macro F_1, and weighted F_1 scores by 0.032, 0.04, and 0.04, respectively, indicating that IPSO's adaptive mechanism outperforms standard PSO. (2) When comparing the full model with "V2", the full model's macro F_1 and weighted F_1 increased by 0.03 and 0.02, respectively. This demonstrates that masking operations help the LSTM model uncover latent features in the data, address the sparsity of raw data, and enhance the model's robustness in extracting target features. (3) When comparing the full model with "V3", the full model achieved varying degrees of superiority in precision, macro F_1, and weighted F_1. Although the

convergence iterations increased, this represents the training cost incurred by IPSO's optimization strategy to avoid premature convergence into local optima.

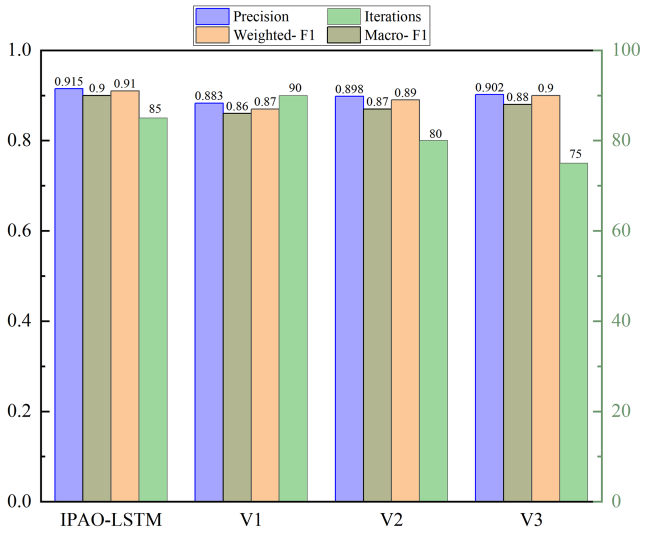


Figure 7: Comparison results of the ablation experiment.

4.3.3 Performance of the IPSO-LSTM

It should be specifically noted that, our enhancements to PSO are not isolated modifications but part of an integrated strategy encompassing: (a) multi-linear decaying inertia weights to improve convergence, (b) adaptive acceleration coefficients, and (c) a novel fitness function that jointly optimizes accuracy and computational efficiency—critical considerations for IoT edge deployments. In the aforementioned experiments, the effectiveness of the proposed method in enhancing the LSTM model has been demonstrated. This section will further validate the superiority of the proposed method in the task of malware detection. In this paper, the penalty factor is set to 0.001, the data of Fig. 6 is input into the LSTM model, the model hyperparameters are optimized using PSO and IPSO under the same initial parameter conditions, with both optimizers' convergence profiles rigorously evaluated to confirm performance gains, as empirically demonstrated in Fig. 8.

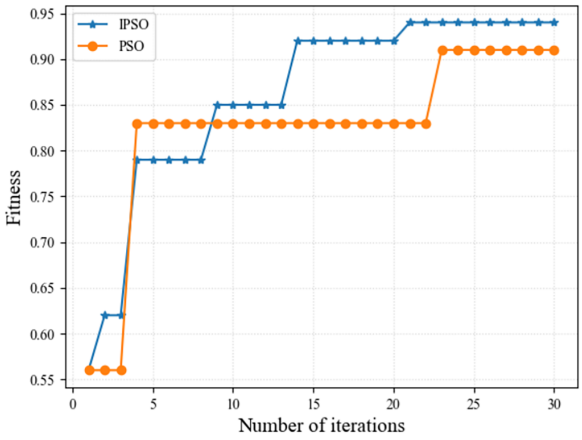


Figure 8: Comparison chart of searching for excellence.

As can be seen in Fig. 8, the IPSO algorithm outperforms the PSO algorithm both in terms of efficiency of optimization search or getting rid of local optima. In 30 iterations, the IPSO algorithm gets rid of the local optimum five times in total, which is better than PSO's two times.

In addition, the final fitness of IPSO is 0.94, which is better than PSO's 0.92. It can be seen that in a limited number of iterations, IPSO has a good optimality finding performance compared to PSO.

Fig. 9 delineates the malware classification efficacy of our engineered IPSO-LSTM framework evaluated on the Mal-API-2019 cybersecurity corpus. In the paper, a standardized data partition (an 8:2 split) is employed, where the graphical outputs present the test set's confusion matrix. From the experimental results, it is easy to see that the proposed model can accurately recognize 8 types of malwares, and there is only a subtle degree of omission and misdetection in different malware recognition tasks. Among them, Trojan has the highest recognition efficiency, with the number of false detections being 3 and the number of missed detections being 10; the rest of the malware's recognition accuracy rate is also above 90%. Since two different types of malware may share similar covert network communication patterns within API sequences, this can lead to confusion during model detection. Furthermore, the core of IPSO-LSTM is temporal modeling, which overlooks statistical features of API sequences (such as call frequency). When the length of malicious API sequences exceeds the memory window of the LSTM, it results in both false detections and missed detections.

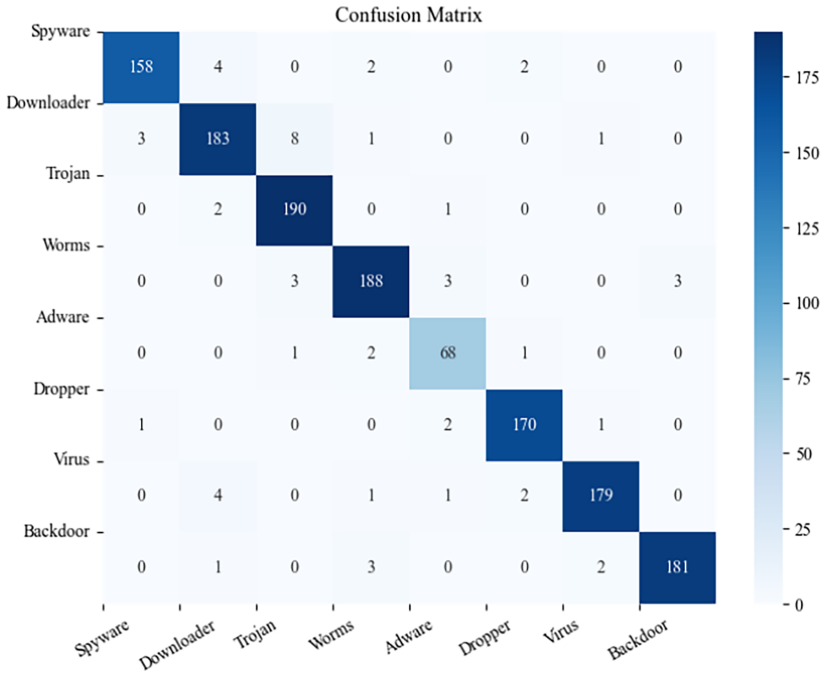
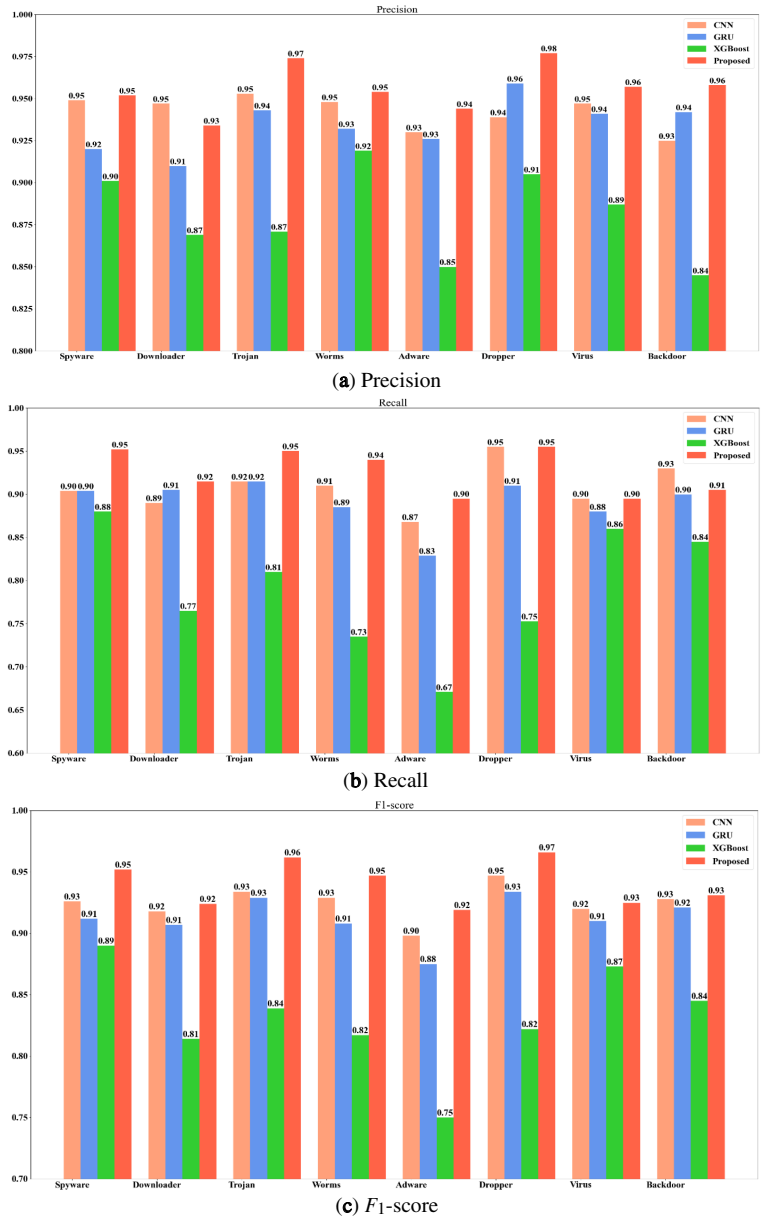


Figure 9: Confusion matrix diagram for the test set.

In the concluding validation phase, comparative trials with state-of-the-art methods are systematically performed to confirm the effectiveness of our developed approach. Ref. [36] constructs a system call vocabulary and uses convolutional neural networks to train pre-labeled data for the malware identification task; Ref. [37] proposes a scalable deep learning-based detection mechanism that effectively identifies multiple types of malware attacks using gated loop units; Ref. [38] converts malware classification into a decision tree classification task and uses malware data to train XGBoost model. Based on the three types of evaluation metrics proposed in Part B, Fig. 10 presents the results from the comparison experiments of

various malware detection models. Here, subgraph a compares the precision of detection for the eight types of malwares; subgraph b compares the recall of detection for the eight types of malwares; and subgraph c compares the F_1 scores for the detection of the eight types of malwares. From the three metrics of Precision, Recall, and F_1-score in the figure, it is easy to conclude that the proposed method is better than the remaining three methods in recognizing the eight types of malwares. In particular, the F_1-score of the IPSO-LSTM malware detection results is above 0.92, and Dropper even reaches 0.97. Further, with other algorithmic projects, IPSO is introduced in the paper to optimize the structure of the LSTM network, which improves the applicability of the model in malware detection, thus improving the key features identified in the malware API call sequences, and realizing the effective identification of 8 types of malwares.



(**a**) Precision

(**b**) Recall

(**c**) F_1-score

Figure 10: Comparison results of different models.

5 Conclusion

In this paper, we propose a new solution to optimize LSTM using IPSO for malware detection. The method innovatively adopts a multilinear decreasing weight strategy to improve the convergence of traditional particle swarm algorithms, finds the globally optimal solution quickly and accurately. The improved particle swarm algorithm optimizes the hyperparameters of the LSTM network, which enhances the performance of the model in malware detection. In order to verify the effectiveness of the method, this paper successfully constructs a stable and efficient malware detection model by training and testing the IPSO-LSTM model with a public dataset, and the experimental results show that the model has excellent detection performance, which provides a strong guarantee for the safe operation of the power mobile terminal.

Although the solution proposed in this paper has achieved significant results in malware detection, there are still some limitations. First, the choice of dataset may affect the generalization ability of the model. Although the public dataset used is representative, the model may not be able to cope with new types of attacks as the variety of malware increases. For this reason, techniques such as generative adversarial networks can be explored to generate more diverse malware samples and enrich the training dataset. Second, the experiments utilized a Windows malware dataset. Direct application to non-Windows, resource-constrained IoT firmware requires further engineering design to achieve low-resource or energy-efficient deployment. Third, further development of lightweight model variants can enable true edge deployment and testing on non-Windows, IoT-specific malware datasets (e.g., those based on ARM system calls), ultimately achieving online detection through real-time processing of streaming API call logs.

Acknowledgement: Not applicable.

Funding Statement: This research was funded by the Science and Technology Project of Yunnan Power Grid Co., Ltd., grant number YNKJXM20222248.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Zheng Yang and Hua Zhu; methodology, Zheng Yang; software, Hua Zhu; validation, Zhao Li, Meng Su and Gang Wang; formal analysis, Zheng Yang; investigation, Zheng Yang; data curation, Hua Zhu; writing—original draft preparation, Zheng Yang; writing—review and editing, Hua Zhu. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the Corresponding Author, Hua Zhu, upon reasonable request.

Ethics Approval: Not applicable.

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

Abbreviations

IoT	Internet of Things
API	Application Programming Interface
LSTM	Long short-term memory network
IPSO	Improved particle swarm optimized

References

1. Jiang A, Wei H, Deng J, Qin H. Cloud-edge cooperative model and closed-loop control strategy for the price response of large-scale air conditioners considering data packet dropouts. IEEE Trans Smart Grid. 2020;11(5):4201–11. doi:10.1109/tsg.2020.2985741.
2. Clarke T, Slay T, Eustis C, Bass RB. Aggregation of residential water heaters for peak shifting and frequency response services. IEEE Open J Power Energy. 2020;7:22–30. doi:10.1109/oajpe.2019.2952804.

3. Yang Y, Wang W, Qin J, Wang M, Ma Q, Zhong Y. Review of vehicle to grid integration to support power grid security. Energy Rep. 2024;12(7):2786–800. doi:10.1016/j.egyr.2024.08.069.
4. Kimani K, Oduol V, Langat K. Cyber security challenges for IoT-based smart grid networks. Int J Crit Infrastruct Prot. 2019;25:36–49. doi:10.1016/j.ijcip.2019.01.001.
5. Libicki MC. Correlations between cyberspace attacks and kinetic attacks. In: 2020 12th International Conference on Cyber Conflict (CyCon); 2020 May 26–29; Tallinn, Estonia. p. 199–213. doi:10.23919/cycon49761.2020.9131731.
6. Goudarzi A, Ghayoor F, Waseem M, Fahad S, Traore I. A survey on IoT-enabled smart grids: emerging, applications, challenges, and outlook. Energies. 2022;15(19):6984. doi:10.3390/en15196984.
7. Ahmad F, Kurugollu F, Kerrache CA, Sezer S, Liu L. NOTRINO: a NOvel hybrid TRust management scheme for INternet-of-vehicles. IEEE Trans Veh Technol. 2021;70(9):9244–57. doi:10.1109/tvt.2021.3049189.
8. Al Shahrani AM, Rizwan A, Sánchez-Chero M, Cornejo LLC, Shabaz M. Blockchain-enabled federated learning for prevention of power terminals threats in IoT environment using edge zero-trust model. J Supercomput. 2024;80(6):7849–75. doi:10.1007/s11227-023-05763-6.
9. Ahmad F, Adnane A, Kurugollu F, Hussain R. A comparative analysis of trust models for safety applications in IoT-enabled vehicular networks. In: 2019 Wireless Days (WD); 2019 Apr 24–26; Manchester, UK. p. 1–8. doi:10.1109/wd.2019.8734204.
10. Ahmad Awan K, Ud Din I, Almogren A, Guizani M, Altameem A, Jadoon SU. RobustTrust—a pro-privacy robust distributed trust management mechanism for Internet of Things. IEEE Access. 2019;7:62095–106. doi:10.1109/ACCESS.2019.2916340.
11. Murugan G, Vijayarajan S. IoT based secured data monitoring system for renewable energy fed micro grid system. Sustain Energy Technol Assess. 2023;57(3):103244. doi:10.1016/j.seta.2023.103244.
12. Ravi N, Shalinie SM. Learning-driven detection and mitigation of DDoS attack in IoT via SDN-cloud architecture. IEEE Internet Things J. 2020;7(4):3559–70. doi:10.1109/JIOT.2020.2973176.
13. Nguyen TG, Phan TV, Hoang DT, Nguyen TN, So-In C. Federated deep reinforcement learning for traffic monitoring in SDN-based IoT networks. IEEE Trans Cogn Commun Netw. 2021;7(4):1048–65. doi:10.1109/tccn.2021.3102971.
14. Liu J, Wang X, Shen S, Yue G, Yu S, Li M. A Bayesian Q-learning game for dependable task offloading against DDoS attacks in sensor edge cloud. IEEE Internet Things J. 2021;8(9):7546–61. doi:10.1109/JIOT.2020.3038554.
15. Abdel-Basset M, Hawash H, Chakrabortty RK, Ryan MJ. Semi-supervised spatiotemporal deep learning for intrusions detection in IoT networks. IEEE Internet Things J. 2021;8(15):12251–65. doi:10.1109/JIOT.2021.3060878.
16. Yan K, Liu X, Lu Y, Yu Z. Thermostatically controlled loads in the power system under cyberattacks. IEEE Internet Things J. 2023;10(6):5256–67. doi:10.1109/JIOT.2022.3222304.
17. Singh NK, Mahajan V. Analysis and evaluation of cyber-attack impact on critical power system infrastructure. Smart Sci. 2021;9(1):1–13. doi:10.1080/23080477.2020.1861502.
18. Babaei Mosleh MR, Sharifian S. An efficient cloud-integrated distributed deep neural network framework for IoT malware classification. Future Gener Comput Syst. 2024;157(7):603–17. doi:10.1016/j.future.2024.03.051.
19. Pan Y, Ge X, Fang C, Fan Y. A systematic literature review of Android malware detection using static analysis. IEEE Access. 2020;8:116363–79. doi:10.1109/ACCESS.2020.3002842.
20. Sihag V, Vardhan M, Singh P. BLADE: robust malware detection against obfuscation in Android. Forensic Sci Int Digit Investig. 2021;38(3):301176. doi:10.1016/j.fsidi.2021.301176.
21. Şahin DÖ, Kural OE, Akleylek S, Kılıç E. Permission-based Android malware analysis by using dimension reduction with PCA and LDA. J Inf Secur Appl. 2021;63(3):102995. doi:10.1016/j.jisa.2021.102995.
22. Tang J, Li R, Jiang Y, Gu X, Li Y. Android malware obfuscation variants detection method based on multi-granularity opcode features. Future Gener Comput Syst. 2022;129(6):141–51. doi:10.1016/j.future.2021.11.005.
23. Alzubi JA, Alzubi OA, Qiqieh I. A feature-learning-enabled malware analysis for enhanced IoT-centric cybersecurity. Clust Comput. 2025;28(13):874. doi:10.1007/s10586-025-05549-w.
24. Wang H, Cui B, Yuan Q, Shi R, Huang M. A review of deep learning based malware detection techniques. Neurocomputing. 2024;598(6):128010. doi:10.1016/j.neucom.2024.128010.

25. Casolare R, De Dominicis C, Iadarola G, Martinelli F, Mercaldo F, Santone A. Dynamic mobile malware detection through system call-based image representation. J Wirel Mob Netw Ubiquitous Comput Dependable Appl. 2021;12(1):44–63. doi:10.22667/JOWUA.2021.03.31.044.
26. Sihag V, Vardhan M, Singh P, Choudhary G, Son S. De-LADY: deep learning based android malware detection using dynamic features. J Internet Serv Inf Secur. 2021;11(2):34–45.
27. Bai H, Xie N, Di X, Ye Q. FAMD: a fast multifeature Android malware detection framework, design, and implementation. IEEE Access. 2020;8:194729–40. doi:10.1109/ACCESS.2020.3033026.
28. Gyamfi NK, Goranin N, Ceponis D, Čenys HA. Automated system-level malware detection using machine learning: a comprehensive review. Appl Sci. 2023;13(21):11908. doi:10.3390/app132111908.
29. Khalis S, Hassini K, Chemmakha M, Habibi O, Lazaar M. Robust CNN-based threat detection in SDN-enabled IoT networks. J Netw Syst Manag. 2026;34(2):40. doi:10.1007/s10922-026-10034-9.
30. Alzubi OA. BotNet attack detection using MALO-based XGBoost model in IoT environment. In: Proceedings of Third International Conference on Computing and Communication Networks; 2023 Nov 17–18; Manchester, UK. p. 679–90. doi:10.1007/978-981-97-2671-4_50.
31. Hou R, Tian X, Geng G. A malware detection method based on LLM to mine semantics of API. EAI Endorsed Trans AI Robotics. 2025;4. doi:10.4108/airo.8880.
32. Lu T, Du Y, Ouyang L, Chen Q, Wang X. Android malware detection based on a hybrid deep learning model. Secur Commun Netw. 2020;2020(6):8863617. doi:10.1155/2020/8863617.
33. Lian W, Nie G, Kang Y, Jia B, Zhang Y. Cryptomining malware detection based on edge computing-oriented multi-modal features deep learning. China Commun. 2022;19(2):174–85. doi:10.23919/JCC.2022.02.014.
34. Xing X, Jin X, Elahi H, Jiang H, Wang G. A malware detection approach using autoencoder in deep learning. IEEE Access. 2022;10:25696–706. doi:10.1109/access.2022.3155695.
35. Anthony Jnr B, Abbas Petersen S, Ahlers D, Krogstie J. API deployment for big data management towards sustainable energy prosumption in smart cities-a layered architecture perspective. Int J Sustain Energy. 2020;39(3):263–89. doi:10.1080/14786451.2019.1684287.
36. Zhang S, Gao M, Wang L, Xu S, Shao W, Kuang R. A malware-detection method using deep learning to fully extract API sequence features. Electronics. 2025;14(1):167.
37. Martinelli F, Marulli F, Mercaldo F. Evaluating convolutional neural network for effective mobile malware detection. Procedia Comput Sci. 2017;112(3):2372–81. doi:10.1016/j.procs.2017.08.216.
38. Bibi I, Akhunzada A, Malik J, Iqbal J, Musaddiq A, Kim S. A dynamic DL-driven architecture to combat sophisticated Android malware. IEEE Access. 2020;8:129600–12. doi:10.1109/ACCESS.2020.3009819.
39. Fereidooni H, Conti M, Yao D, Sperduti A. ANASTASIA: ANdroid mAlware detection using STatic analySIs of applications. In: 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS); 2016 Nov 21–23; Larnaca, Cyprus. p. 1–5. doi:10.1109/NTMS.2016.7792435.
40. Catak FO, Yazı AF, Elezaj O, Ahmed J. Deep learning based Sequential model for malware analysis using Windows exe API Calls. PeerJ Comput Sci. 2020;6(81):e285. doi:10.7717/peerj-cs.285.
41. Golmaryami M, Taheri R, Pooranian Z, Shojafar M, Xiao P. SETTI: a Self-supervised advErsarial malware deTection archiTecture in an IoT environment. ACM Trans Multimedia Comput Commun Appl. 2022;18(2s):1–21. doi:10.1145/3536425.