Journal of Network Intelligence
Taiwan Ubiquitous Information
 ISSN 2414-8105
Volume **11**, Number **1**, February **2026**

Intelligent Detection of Malicious APP Software Based on Fuzzy Neural Networks

Feng-Fan Xin[1,*]

[1]School of Information Engineering,
Zhengzhou College of Finance and Economics, Zhengzhou 450000, P. R. China
xinfengfan130@163.com

Dong-Hua Zheng[2]

[2]Graduate School,
Management and Science University, Shah Alam 40100, Selangor, Malaysia
20190257@gcc.edu.cn

[*]Corresponding author: Feng-Fan Xin

Received November 18, 2024, revised February 14, 2025, accepted May 23, 2025.

Abstract. *Traditional malicious app detection methods usually rely on static analysis or fixed machine learning models, and lack real-time responsiveness to dynamically changing and emerging threats. With the rapid development of the mobile app ecosystem, there is an increasing demand for detection systems that can monitor, process large amounts of data, and perform intelligent analysis in real-time. To solve this problem, this paper proposes a malicious APP detection system based on Fuzzy Neural Network (FNN) and adaptive optimization algorithm. First, the system collects and processes the behavioral features of APPs in real time through Fuzzy Logic Module (FLM) and Deep Neural Network (DNN), and uploads the data to the cloud platform for storage and analysis. Secondly, the combination of Online Learning (OL) and Reinforcement Learning (RL) technologies enables the dynamic adjustment of fuzzy rules and neural network weights, thus enhancing the system's adaptability to new and evolving malicious behaviors. The experiments are conducted on two public datasets, AndroZoo and Drebin, and compared with five benchmark methods, the method proposed in this paper improves the classification accuracy by 5%, the adaptation detection rate by 11%, and the computational efficiency by 30%. The experimental results show that the method in this paper outperforms the existing methods in terms of accuracy, adaptability and computational efficiency, and has high potential for application.*
Keywords: malicious app detection; fuzzy neural network; adaptive optimization; online learning; reinforcement learning

1. **Introduction.** With the widespread popularization of smartphones and mobile applications, mobile applications (APPs) have become an indispensable part of users' daily lives. However, along with the rapid expansion of the app ecosystem, the number and complexity of malicious APPs are increasing, posing serious threats to user privacy and data security [1, 2]. Malicious APPs may not only steal users' sensitive information, but also abuse system resources, leading to device performance degradation or even damage [3, 4]. Therefore, the development of efficient and intelligent malicious APP detection methods has become an important issue for ensuring mobile security. Traditional static and dynamic analysis methods show certain limitations in the face of increasingly complex

and hidden malicious behaviors, and there is an urgent need to introduce more adaptive and robust technical means.

1.1. **Related work.** Traditional malicious app detection methods rely on static analysis [5] and dynamic analysis techniques [6]. Static analysis methods identify potentially malicious behaviors by examining features such as the code structure, permission requests, and signature information of the application. For example, Talha et al. [7] proposed a static detection method based on permission auditing, which analyzes the set of permissions requested by an app to determine whether it is a potential malicious risk. This method performs well in detecting known malicious permissions, but has limited ability to identify new malicious APPs with permission abuse. In addition, Li et al. [8] developed a static analysis tool based on control-flow graphs to identify malicious behaviors by comparing the control-flow characteristics of an app with templates in a malicious sample library. Although this method improves the accuracy of detection to some extent, its effectiveness decreases significantly when facing malicious APPs with obfuscation and code hiding techniques.

With the development of machine learning and deep learning technology, intelligent detection methods have become the mainstream direction of malicious APP identification. Chen et al. [9] utilized a support vector machine (SVM) model to achieve efficient malicious APP classification by extracting static and dynamic features of APPs. The study shows that SVM has high accuracy in processing high-dimensional feature data, but its adaptability to new malicious samples is insufficient. Another study, proposed by Yan et al. [10], used deep neural networks (DNN) to model APP behaviors and achieved recognition of complex malicious patterns. Although DNN performs well in detecting complex malicious behaviors, its model training process consumes large computational resources and is difficult to apply in real-time. In addition, Urooj et al. [11] introduced an integrated learning approach to improve the overall performance of malicious APP detection by combining the prediction results of multiple classifiers. However, the integration approach faces the challenges of high model complexity and high maintenance cost in practical deployment, which limits the possibility of its wide application.

Fuzzy logic, as an effective tool for dealing with uncertainty and fuzzy information, has shown some application prospects in the field of information security in recent years. Rodriguez-Bazan et al. [12] explored the application of fuzzy neural networks in network intrusion detection, and improved the ability to recognize complex attack patterns by fusing fuzzy rules and neural network models. However, the study mainly focused on the network environment, and the specific requirements for malicious APP detection have not been fully considered, and there is the problem of insufficient model generalization. Another study, according to Kuk et al. [13], attempts to apply fuzzy logic to malware classification, and realizes fine-grained classification of malicious behaviors by processing multi-dimensional feature data through fuzzy affiliation function. Although this method improves the classification accuracy to a certain extent, its optimization algorithm lacks adaptivity, which makes it difficult to cope with the dynamic changes of malicious APPs and the continuous emergence of new types of threats.

The above literature analysis shows that traditional methods have limitations in dealing with complex and dynamically changing malicious APP behaviors, while current intelligent detection methods, although possessing high accuracy, still need to be improved in terms of adaptability and real-time performance. Meanwhile, fuzzy logic techniques show potential in dealing with uncertainty and complex features.

1.2. **Motivation and contribution.** Existing fuzzy logic techniques show some application potential in the field of malicious APP detection, but there are still many challenges.

Existing studies have explored the application of fuzzy neural networks in network intrusion detection, and although they have improved the ability of recognizing complex attack patterns, they have mainly focused on the network environment, and have not fully considered the specific needs of malicious APP detection, resulting in insufficient model generalization. Researchers try to apply fuzzy logic to malware classification, and achieve fine-grained classification of malicious behaviors by processing multi-dimensional feature data through fuzzy affiliation function, however, its optimization algorithm lacks adaptivity, which makes it difficult to cope with the dynamic changes of malicious APPs and the continuous emergence of new types of threats. Aiming at the above problems, this paper proposes a fuzzy neural network-based intelligent detection method for malicious APP software, and the main innovations and contributions include:

(1) A hybrid architecture integrating fuzzy logic into deep neural networks is designed to effectively deal with the uncertainty and ambiguity in malicious APP behavior. The architecture achieves the refinement and optimization of fuzzy features by introducing a fuzzy rule module in the hidden layer of the neural network, which improves the robustness and classification accuracy of the detection model. This approach solves the problem of insufficient model generalization in the study of Rodriguez-Bazan et al. [12] and enhances the applicability of the model in malicious APP detection.

(2) An adaptive optimization algorithm is proposed that can dynamically adjust the connection weights of fuzzy rules and neural networks according to the emergence of emerging threats. The algorithm adopts online learning mechanism and reinforcement learning techniques to achieve real-time adaptation and optimization of the fuzzy neural network in the face of the evolution of malicious APP behaviors. This innovation effectively solves the problem of the optimization algorithm's lack of adaptivity in the study of Kuk et al. [13], and significantly improves the detection system's ability to cope with dynamic changes and novel threats.

2. Hybrid Architecture Incorporating Fuzzy Logic and Deep Neural Networks.

2.1. Fuzzy logic module design.

In the complex environment of malicious app detection, application behavior data usually contains a high degree of uncertainty and ambiguity. Traditional deterministic models are difficult to effectively capture the intrinsic relationships of these features, so it is particularly important to introduce a fuzzy logic module to enhance the model's ability to handle uncertainty and ambiguity. Designing an efficient fuzzy logic module can not only enhance the robustness of the detection system, but also improve the classification accuracy. In this section, the structural design of the fuzzy logic module, the definition of fuzzy rules, and the selection method of the affiliation function will be elaborated in detail, so as to realize the accurate fuzzification of malicious APP features.

Let the input feature vector be $\mathbf{X} = [x_1, x_2, \ldots, x_n]$, where x_i denotes the i-th feature. Each feature x_i is mapped to the corresponding fuzzy set F_i through the affiliation function $\mu_i(x_i)$, i.e.

$$F_i = \mu_i(x_i) \tag{1}$$

where $\mu_i(x_i)$ is defined in the $[0, 1]$ interval of the affiliation function, often in the form of a Gaussian function, triangular function or trapezoidal function. For example, the Gaussian generating function $\mu_i(x_i)$ has the form of

$$\mu_i(x_i) = e^{-\frac{(x_i - c_i)^2}{2\sigma_i^2}} \tag{2}$$

where c_i is the center value of the affiliation function and σ_i is the standard deviation, which determines the width of the affiliation function. Choosing the appropriate form of

the affiliation function depends on the distribution characteristics of the features and the detection needs to ensure that the ambiguity of the features can be captured effectively.

The definition of fuzzy rules is another key aspect of fuzzy logic module design. A fuzzy rule describes the logical relationship between features in the form of "if-then". For example, the rule R_j can be expressed as:

$$R_j : \text{if } F_{j1} \text{ and } F_{j2}, \text{ then } F_{j\text{out}} \tag{3}$$

where F_{j1} and F_{j2} correspond to the fuzzy sets of the input features respectively; $F_{j\text{out}}$ is the output fuzzy set. The fuzzy inference process matches the input fuzzy sets with the rules through logical operators (e.g., "with", "or") to generate the fuzzified output. Specifically, the activation level α_j of the rule R_j is computed through a minimization operation:

$$\alpha_j = \min(\mu_{j1}(x_{j1}), \mu_{j2}(x_{j2})) \tag{4}$$

The outputs of all the activation rules are aggregated by the weighted average method to obtain the combined fuzzy output Y:

$$Y = \frac{\sum_{j=1}^{m} \alpha_j \cdot y_j}{\sum_{j=1}^{m} \alpha_j} \tag{5}$$

where y_j denotes the output value corresponding to the rule R_j. In order to achieve seamless integration of the fuzzy logic module with the DNN, an end-to-end design approach is used. The fuzzy logic module acts as a hidden layer of the DNN [14, 15], allowing the fuzzified features to be propagated and optimized in the network. Specifically, the fuzzified output Y is passed to the next layer of the DNN for further feature extraction and pattern recognition. The framework of the fuzzy logic module is shown in Figure 1.

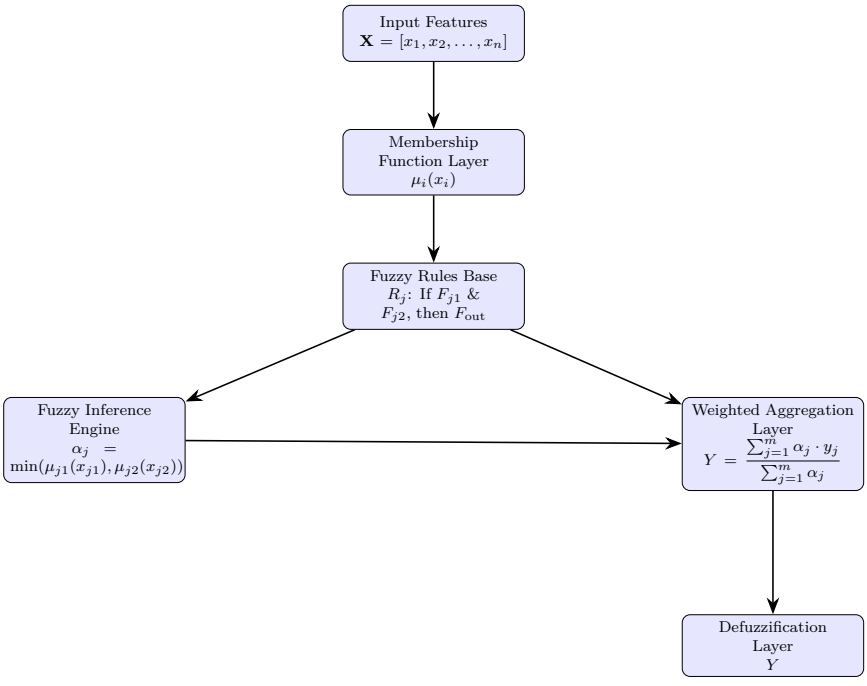

Figure 1. Framework diagram of the fuzzy logic module

2.2. **Deep neural network integration.** Based on the design of the aforementioned fuzzy logic module, this study further integrates it seamlessly into the DNN to enhance the performance and adaptability of the overall detection system. This integration process not only retains the advantages of DNN in feature learning and pattern recognition, but also gives full play to the advantages of the fuzzy logic module in dealing with uncertainty and complex data, realizing the complementary advantages of the two.

In this integrated architecture, the fuzzy logic module acts as the first hidden layer of the DNN and receives the fuzzified output Y from the input feature vector $\mathbf{X} = [x_1, x_2, \ldots, x_n]$. Specifically, the input features are first processed by the fuzzy logic module to generate a comprehensive fuzzy output Y, after which Y is passed to the subsequent hidden layers of the DNN for deeper feature extraction and pattern recognition. Such a design not only retains the powerful capability of traditional DNN in feature learning and representation, but also incorporates the advantages of fuzzy logic module in handling uncertain data and rule-based reasoning, realizing the complementary advantages of the two.

The output Y of the fuzzy logic module is further processed as new input features when passed to the next layer of the DNN. While traditional DNNs usually rely on a large amount of data and complex nonlinear transformations to learn the relationships between features, the introduction of the fuzzy logic module enables the network to capture the fuzzy and uncertain information in the data at an early stage, thus improving the overall network learning efficiency and classification accuracy. In this way, the fuzzy logic module not only optimizes the feature representation, but also provides richer and finer input information to the DNN, which enhances the model's ability to discriminate malicious APP behaviors.

During the training process, the whole integrated model adopts an end-to-end training strategy to simultaneously optimize the parameters θ_{Fuzzy} of the fuzzy logic module and the weights θ_{DNN} of the DNN through the back-propagation algorithm. The total loss function $\mathcal{L}$ consists of the main loss term of the DNN and the auxiliary loss term of the fuzzy logic module, namely

$$\mathcal{L} = \mathcal{L}_{\text{DNN}} + \lambda \mathcal{L}_{\text{Fuzzy}} \tag{6}$$

where $\mathcal{L}_{\text{DNN}}$ denotes the main loss term (e.g., cross-entropy loss) of the DNN; $\mathcal{L}_{\text{Fuzzy}}$ denotes the loss term of the fuzzy logic module; and λ is the weight parameter, which is used to balance the contributions of the two parts. By minimizing $\mathcal{L}$, the training process not only optimizes the performance of the DNN in the classification task, but also adjusts the rule adaptation of the fuzzy logic module so that it can dynamically respond to changes in the data distribution. This joint optimization mechanism ensures the simultaneous update of the fuzzy logic module and DNN weights, which improves the robustness and accuracy of the overall detection system.

Through backpropagation, the gradient $\partial\mathcal{L}/\partial\theta$ is computed and used to simultaneously update the DNN weights θ_{DNN} and the fuzzy rule parameters θ_{Fuzzy}, namely

$$\theta_{\text{DNN}}, \theta_{\text{Fuzzy}} \leftarrow \theta_{\text{DNN}}, \theta_{\text{Fuzzy}} - \eta \frac{\partial \mathcal{L}}{\partial \theta} \tag{7}$$

where η is the learning rate. Through this joint optimization mechanism, the fuzzy logic module and DNN can work together and enhance each other's advantages, thus significantly improving the overall performance of the malicious APP detection system.

In addition, a multi-layer fuzzy affiliation function design is introduced to enhance the expressive power of the fuzzy logic module [16, 17]. By applying different granularities of fuzzification at different levels, it is able to capture the multi-scale information of the features and further refine the behavioral patterns of the malicious APP. Let the k-th

level fuzzy affiliation function be $\mu_i^{(k)}(x_i)$, then the multilayer fuzzification output $F_i^{(k)}$ can be expressed as:

$$F_i^{(k)} = \mu_i^{(k)}(x_i) \tag{8}$$

By applying different granularity of affiliation functions at different levels, the model is able to comprehensively utilize the information at multiple levels to improve the recognition ability of malicious behaviors. This multi-layer fuzzification not only improves the model's ability to model complex feature relationships, but also enhances the system's adaptability in the face of new types of malicious behaviors, ensuring that it can still maintain efficient detection performance in dynamic threat environments.

Through the integrated design of the above deep neural network and fuzzy logic module, this paper realizes a malicious APP detection method with both high accuracy and strong adaptability. This method effectively combines the advantages of fuzzy logic in dealing with uncertain data with the powerful capabilities of DNN in feature learning and pattern recognition, which significantly improves the overall performance of the detection system in the face of complex and dynamically changing malicious APP threats. This innovative architecture not only provides new ideas for the design of intelligent security systems, but also lays a solid foundation for future related research.

3. Adaptive fuzzy neural network optimization algorithm.

3.1. Online learning mechanism design.

In order to enhance the adaptability and responsiveness of the malicious APP detection system in the face of dynamic changes and emerging threats, this paper designs an adaptive optimization mechanism based on Online Learning (OL) [18, 19] and Reinforcement Learning (RL) [20, 21]. The mechanism aims to dynamically adjust the parameters of the Fuzzy Logic Module (FLM) and Deep Neural Network (DNN) by processing the newly arrived data in real time, thus significantly enhancing the accuracy and robustness of the system in detecting unknown threats.

The core idea of the online learning mechanism is to utilize an incremental updating strategy that allows the model to instantly adjust its parameters as it receives new data without retraining the entire network. Specifically, located at time step t, the system receives a new input feature vector $\mathbf{X}_t = [x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}]$. The fuzzy logic module first fuzzifies $\mathbf{X}_t$ to generate the fuzzy output Y_t, which is then passed to the hidden layer of the DNN for further feature extraction and pattern recognition. The parameter update of this process follows the following equation:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_t \tag{9}$$

where $\theta = [\theta_{\text{DNN}}, \theta_{\text{FLM}}]$ denotes all trainable parameters of DNN and FLM; η is the learning rate, which controls the pace of parameter updating; $\nabla_\theta \mathcal{L}_t$ denotes the gradient of the total loss function $\mathcal{L}_t$ with respect to the parameter θ at time step t.

3.2. Application of reinforcement learning in optimization.

In order to further enhance the adaptive capability of the model, this paper incorporates the policy optimization method of RL into the online learning mechanism. Reinforcement learning, as a machine learning method that acquires optimal policies by interacting with the environment, is able to autonomously learn and optimize the decision-making process in uncertain and dynamically changing environments. In this study, the RL agent is designed to dynamically adjust the fuzzy rule parameters θ_{FLM} and DNN weights θ_{DNN} to maximize the overall performance of the detection system. The RL agent receives environmental feedback signals by interacting with the malicious APP behavior detection environment r_t, and adjusts the parameter update strategy based on these feedback signals to optimize the detection capability of the model. Specifically, the RL agent (Agent) adjusts

the parameter update strategy based on the feedback signals r_t of the current detection results to maximize the long-term cumulative rewards R. The reward signal r_t reflects the accuracy of the current detection result and the overall performance of the system, which is defined as follows:

$$R = \sum_{t=1}^{T} \gamma^{t-1} r_t \tag{10}$$

where T is the total number of training rounds and γ is a discount factor to measure the importance of future rewards. The RL agent adapts to changes in emerging threats by optimizing its strategy so that the parameter update process can maximize R while adapting to changes in emerging threats.

At each time step t, the RL agent selects an action a_t based on the current state s_t (including the current parameter settings and detection results), which defines the direction and magnitude of the parameter update. The specific update formula is:

$$\theta \leftarrow \theta + \eta \cdot a_t \tag{11}$$

where η is the learning rate to control the pace of parameter update. Through continuous trial and optimization, the RL agent can gradually approach the optimal strategy and realize the intelligent adjustment of parameters.

In order to realize the collaborative work between the RL agent and the online learning mechanism, a joint optimization strategy is designed in this study. At each time step t, the system first processes the newly arrived data $\mathbf{X}_t$ through the online learning mechanism, generates the fuzzified output $\mathbf{Y}_t$ and performs forward propagation to compute the classification result and loss $\mathcal{L}_t$. Subsequently, based on the feedback signal r_t of the classification result, the RL agent evaluates the effect of the current parameter settings and decides whether and how to adjust the parameter update strategy.

The action a_t of the RL agent not only affects the pace of parameter updates, but also determines the direction of parameter updates to ensure that parameter adjustments are made in the direction of improving detection performance. The RL agent optimizes the parameter update strategy by maximizing the cumulative rewards R to improve the system's detection accuracy and robustness in the face of new and evolving threats. The optimization process of the parameter update strategy under the guidance of reinforcement learning is as follows:

$$\theta \leftarrow \theta - \eta \left(\nabla_\theta \mathcal{L}_t + \beta \nabla_\theta Q(s_t, a_t; \phi) \right) \tag{12}$$

where β is a weight parameter regulating the influence of RL component in parameter update, $Q(s_t, a_t; \phi)$ is the value function of action a_t under state s_t, and the parameter ϕ is the parameter of strategy network. Through joint optimization, the model is able to dynamically adjust the fuzzy rules of FLM while maintaining the classification performance of DNN to enhance the overall system's ability to recognize complex and evolving malicious behaviors.

In addition, in order to prevent Data Drift [22] and Concept Drift [23] that may occur during online learning, this paper introduces a data management strategy based on Sliding Window. Setting the sliding window size as w, the system uses only the most recent w data samples for parameter updating. This strategy not only ensures the sensitivity of the model to the latest threats, but also retains the memory of historical data trends, which enhances the generalization ability and stability of the model. The set of data samples within the sliding window is noted as $\mathcal{D}_t = \{\mathbf{X}_{t-w+1}, \mathbf{X}_{t-w+2}, \ldots, \mathbf{X}_t\}$.

At each time step t, the model achieves online learning and optimization through the following steps: First, the latest input data $\mathbf{X}_t$ is extracted from the sliding window and the fuzzy output Y_t is generated. Then, the DNN is utilized to forward propagate Y_t

and calculate the loss $\mathcal{L}_t$ between the prediction result and the true label. Next, the parameter update strategy is adjusted to optimize the parameters θ of FLM and DNN based on the reward signal r_t provided by the RL agent. Finally, the gradient $\nabla_\theta \mathcal{L}_t$ and $\nabla_\theta Q(s_t, a_t; \phi)$ are computed by the back-propagation algorithm and the parameters are updated to minimize the total loss function.

This adaptive optimization mechanism combining online learning and RL enables Fuzzy Neural Network (FNN) to respond to threat changes in real time in dynamic environments and continuously improve detection performance. Meanwhile, the sliding window strategy effectively balances the model's dependence on old and new data, ensuring that the system remains efficient and reliable in the face of changing security threats.

3.3. **Adaptive rule adjustment.** To ensure the effectiveness and stability of fuzzy rule tuning, this paper designs a method based on Policy Gradient (PG). By maximizing the cumulative reward R, the RL agent optimizes the parameters ϕ of its policy network π such that the selected action a_t can maximize the expected reward in the current state s_t. The strategy gradient is updated by:

$$\nabla_\phi J(\phi) = \mathbb{E}_\pi \left[\nabla_\phi \log \pi(a_t \mid s_t; \phi) R_t \right] \tag{13}$$

where $J(\phi)$ is the performance index of the strategy, $\mathbb{E}_\pi$ denotes the expectation under the strategy π, and R_t is the cumulative reward of the time step t. By constantly updating ϕ, the RL agent is able to learn the optimal parameter tuning strategy, thus achieving adaptive optimization of fuzzy rules.

In addition, this paper introduces an Experience Replay mechanism to store and sample historical interaction data to break the correlation between data and improve the efficiency and stability of policy learning. The agent records the state s_t, action a_t and reward r_t at each time step t and stores them in the Experience Replay buffer. During the training process, a small batch of samples are randomly selected from the buffer for policy updating to reduce the bias and variance during the training process.

It is crucial to ensure the consistency of the fuzzy rules and the robustness of the system during the rule tuning process. For this reason, this paper introduces a rule consistency constraint term $\mathcal{L}_{\text{Consistency}}$ into the design of the reward signal r_t, which is used to measure the magnitude of changes before and after the fuzzy rule tuning, and to avoid over-tuning of the rule parameters leading to fluctuations in the system performance. The total reward r_t is calculated as:

$$r_t = \mathcal{R}_{\text{Performance}} - \lambda_{\text{Consistency}} \mathcal{L}_{\text{Consistency}} \tag{14}$$

where $\mathcal{R}_{\text{Performance}}$ denotes the reward signal associated with detection performance; $\lambda_{\text{Consistency}}$ is a weight parameter that regulates the importance of the consistency constraint; $\mathcal{L}_{\text{Consistency}}$ is defined as the amount of variation in the fuzzy rule parameters between time steps $t-1$ and t:

$$\mathcal{L}_{\text{Consistency}} = \|\theta_{\text{FLM}}^{(t)} - \theta_{\text{FLM}}^{(t-1)}\|_2 \tag{15}$$

where $\theta_{\text{FLM}}^{(t)}$ denotes the fuzzy rule parameter of the current time step t. Through this design system, while optimizing the detection performance, the system maintains the smoothness of the fuzzy rule adjustment, prevents the negative impact of over-adjustment, and ensures the stable operation of the detection system in various complex environments.

The pseudo-code of the proposed adaptive fuzzy neural network optimization algorithm is shown in Algorithm 1.

First, the initial values of fuzzy rule parameters and DNN weights are set, and the RL strategy parameters are initialized, as well as the empirical playback buffer and sliding window to manage the data. The system receives new input feature vectors, adds them to the sliding window, and generates fuzzy outputs by performing fuzzification through

Algorithm 1 Pseudo-code for calculating the distance between whale individuals

Input: Initial fuzzy rule parameters $\theta_{\text{FLM}}^{(0)}$, Initial deep neural network weights $\theta_{\text{DNN}}^{(0)}$, learning rate η, weights λ, sliding window size w, iterations total number T.

Output: Optimized fuzzy rule parameters $\theta_{\text{FR}}^{(T)}$, Optimized deep neural network weights $\theta_{\text{DNN}}^{(T)}$.

begin
$\theta_{\text{FR}} \leftarrow \theta_{\text{FR}}^{(0)}$, $\theta_{\text{DNN}} \leftarrow \theta_{\text{DNN}}^{(0)}$;
Initialize RL policy parameters ϕ;
Initialize the experience playback buffer $\mathcal{B} \leftarrow \varnothing$;
Initialize the sliding window $\mathcal{D} \leftarrow \varnothing$
for $t = 1$ **to** T **do**
  Receive new input feature vector $\mathbf{X}_t = [x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}]$;
  Add $\mathbf{X}_t$ to the sliding window $\mathcal{D}$;
  The FLM fuzzifies $\mathbf{X}_t$ to generate the fuzzy output Y_t;
  The DNN forward propagates Y_t to obtain the predicted output $\hat{y}_t$;
  Calculate the total loss function $\mathcal{L}_t = \mathcal{L}_{\text{DNN}}(\hat{y}_t, y_t) + \lambda \mathcal{L}_{\text{FLM}}(Y_t)$;
  Calculation of the gradient $\nabla_\theta \mathcal{L}_t$ of the loss function with respect to the parameters;
  The RL agent (Agent) selects action a_t based on the current state s_t;
  $\theta \leftarrow \theta - \eta(\nabla_\theta \mathcal{L}_t + \beta \nabla_\theta Q(s_t, a_t; \phi))$;
  Calculating the reward signal r_t;
  Store experience (s_t, a_t, r_t, s_{t+1}) to buffer $\mathcal{B}$;
  If the buffer $\mathcal{B}$ is full **then**
  a small batch of samples is randomly drawn (s_i, a_i, r_i, s_{i+1});
  end if
  Computing strategy gradients $\nabla_\phi J(\phi)$ using small sample sizes;
  Updating RL policy parameters $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$;
  If the size of the sliding window $\mathcal{D}$ exceeds w **then**
    the oldest sample is removed;
  end if
end for
return $\theta_{\text{FR}}^{(T)}$, $\theta_{\text{DNN}}^{(T)}$
end

FLM. DNN performs forward propagation of the fuzzy outputs to obtain the prediction results and calculates the total loss function, which combines the classification loss of the DNN with the auxiliary loss of the FLM.

The gradient of the loss function with respect to the model parameters is computed by the back propagation algorithm, and the RL agent selects an action based on the current state and updates the fuzzy rule parameters and DNN weights with the gradient information. The system computes the reward signal for the current step, stores the experience to a buffer, and performs a small batch of samples when the buffer is full for

gradient computation and parameter update of the RL policy. Ensure that only the latest w samples are kept in the sliding window to adapt to the dynamically changing malicious APP behavior patterns.

The optimized fuzzy rule parameters and DNN weights are finally returned to complete the optimization process of adaptive fuzzy neural network. Through this adaptive optimization algorithm, the fuzzy neural network is able to respond to the changes in the behavior of the malicious APP in real time and dynamically adjust the fuzzy rules and the neural network weights, which significantly improves the accuracy and robustness of the detection system.

4. **Experimental design and evaluation.**

4.1. **Dataset and preprocessing.**

In order to systematically evaluate the effectiveness and practicality of the proposed fuzzy neural network-based malicious APP detection method, this paper selects several representative and diverse publicly available malicious APP datasets for experiments. The selected datasets cover different types of malicious behaviors and multiple application scenarios, aiming to comprehensively test the model's generalization ability and adaptability in real-world applications. Table 1 shows the specific parameters of the two public datasets used in this paper.

Table 1. Specific parameters of the data set

Parameters	**AndroZoo**	**Drebin**
Sample size	Over 1,000,000	5,615
Number of malicious samples	About 50,000	5,560
Normal sample size	About 950,000	55
Feature type	Static and dynamic characteristics	Static characteristics
Number of features	Multi-dimensional (permission requests, API call frequency, network connection patterns, etc.)	Multi-dimensional (permission requests, API call frequency, network connection patterns, etc.)
Data sources	Diverse Android Apps	Carefully labeled malicious Android apps
Labels	Malice and normalcy	Malice and normalcy
Data collection time	Continuously updated	2014 collection

AndroZoo dataset [24]: AndroZoo is a comprehensive database of millions of Android apps covering various categories of apps, including normal apps and many types of malicious apps. The dataset provides rich static and dynamic features for large-scale malicious behavior analysis and classification tasks.

Drebin dataset [25]: Drebin is a high-quality dataset focusing on malicious Android apps, containing more than 5,000 carefully labeled malicious app samples. The dataset records in detail the characteristics of malicious APPs in multiple dimensions such as permission requests, component calls, network behaviors, etc., and is one of the commonly used benchmark datasets in malicious behavior detection research.

Prior to model training and evaluation, the selected dataset was systematically preprocessed to ensure data quality and feature consistency. The preprocessing process includes the following steps:

(1) **Feature extraction and selection**: for each APP sample, multi-dimensional behavioral features are extracted, including Permissions, API Call Frequency, Network Connection Patterns and so on. These features can comprehensively reflect the behavioral patterns and potential malicious activities of APPs. In order to reduce the dimensionality disaster and improve the computational efficiency of the model, methods such as correlation analysis and Principal Component Analysis (PCA) are used to filter out the most discriminative subset of features.
(2) **Data cleaning and normalization**: In the process of feature extraction, there are some missing values and outliers. Through the interpolation method and outlier detection mechanism, the missing data were filled in and the abnormal samples were removed to ensure the completeness and representativeness of the dataset. In addition, all numerical features were standardized, using the Z-score standardization method with a mean of 0 and variance of 1 to unify the scale of the features and prevent different scales from adversely affecting the model training.
(3) **Data set division**: to evaluate the generalization ability of the model, a Cross-Validation (CV) strategy was adopted to divide the data set into training, validation and testing sets. Specifically, according to the ratio of 80-10-10, 80% of the data is used for model training, 10% for parameter tuning, and the remaining 10% for the final performance evaluation. Such a division helps to fully utilize the limited data resources and improve the stability and reliability of the model.
(5) **Fuzzification**: given that the model proposed in this paper incorporates FLM, the selected features need to be fuzzified in the data preprocessing stage. Gaussian Membership Function (GMF) is used to fuzzify the continuous type features, mapping each feature value to the corresponding fuzzy set to capture the fuzzy relationship and uncertainty between the features.
(6) **Data Enhancement and Balancing**: Considering the possible category imbalance of malicious APP samples in the dataset, this paper employs Oversampling (Oversampling) and Undersampling (Undersampling) techniques to equalize the distribution of various categories of samples. In addition, the Synthetic Minority Oversampling Technique (SMOTE) is combined to generate synthetic malicious APP samples to further improve the model's ability to recognize minority samples and enhance the comprehensiveness and robustness of the detection system.

4.2. **Benchmark methods and assessment indicators.** In order to comprehensively assess the effectiveness and advantages of the malicious APP detection methods proposed in this paper, five representative benchmark methods are selected for comparative analysis.

In order to comprehensively evaluate the performance of the malicious APP detection method proposed in this paper, a series of evaluation metrics corresponding to the two innovations in this paper are selected. These metrics aim to measure the performance of the detection system in terms of classification accuracy, adaptive capability, and computational efficiency to ensure that the proposed method has significant advantages in dealing with uncertainty and dynamic threat environments.

Classification accuracy is one of the core metrics to measure the performance of the malicious APP detection system, including Accuracy, Precision, Recall and F1-score.

The evaluation of adaptive capability focuses on the system's responsiveness and adaptability to new and evolving malicious APP behaviors. To this end, the following metrics are designed in this paper:

(1) Adaptive Detection Rate (ADR): a measure of the system's ability to quickly recognize new types of malicious samples when they are received. ADR is defined as the proportion of new malicious samples that the system correctly detects within a recent update or new data cycle.

$$\text{ADR} = \frac{\text{Number of correctly detected new malicious samples}}{\text{Total number of new malicious samples}} \tag{16}$$

(2) Update Time (UT): evaluates the time it takes for the system to complete parameter updates after receiving new data. A shorter update time means that the system can respond more quickly to threat changes.

$$\text{UT} = \frac{\text{Total time taken for parameter updates}}{\text{Number of updates}} \tag{17}$$

Computational efficiency reflects the resource consumption and processing speed of the system when performing malicious APP detection, covering the following aspects:

(1) Average Detection Time (ADT): the average detection time consumed for each APP sample, which directly measures the real-time performance of the model in practical applications.

$$\text{ADT} = \frac{\sum_{i=1}^{N} T_i}{N} \tag{18}$$

where T_i is the detection time of the i-th APP sample, and N is the total number of detected samples.

(2) Computational Resource Consumption (CRC): Includes CPU and memory usage, which can reflect the resource requirements of the model when processing large-scale data. It is usually compared by measuring the average CPU utilization and memory usage of each model under the same hardware configuration.

$$\text{CRC} = \frac{\sum_{i=1}^{N} (\text{CPU}_i + \text{Memory}_i)}{N} \tag{19}$$

where CPU_i and Memory_i denote the CPU utilization and memory occupancy at the detection of the i-th sample, respectively.

In this paper, all models are trained and tested under fair conditions, using the same hardware environment and parameter settings. In order to reduce the effect of randomness in the experiment, multiple Cross-Validation (CV) is used to ensure the stability and reliability of the results. In addition, the statistical significance test is used to verify whether the improvement of this paper's method on each index is statistically significant.

4.3. **Experimental results and analysis.**

4.3.1. *Detection accuracy.* As can be seen from Figure 2, this paper's method significantly outperforms the benchmark method in all detection accuracy metrics, with an accuracy of 95%, which is 5% higher than the second place, and the F1-score is also improved by about 5%. This indicates that the method in this paper has higher accuracy and robustness in classifying malicious apps, and can effectively distinguish between malicious and normal apps, and especially performs well in dealing with complex and confusing malicious behavior patterns.

4.3.2. *Model adaptation.* From Table 2, it can be seen that this paper's method is ahead of the benchmark method in both adaptation detection rate (ADR) and update time. The ADR reaches 85%, showing that this paper's method is more adaptive in recognizing novel malicious samples. Meanwhile, the update time is only 35 ms, which is about 10 ms faster than the second-place Kuk et al. [13], indicating that the parameter update

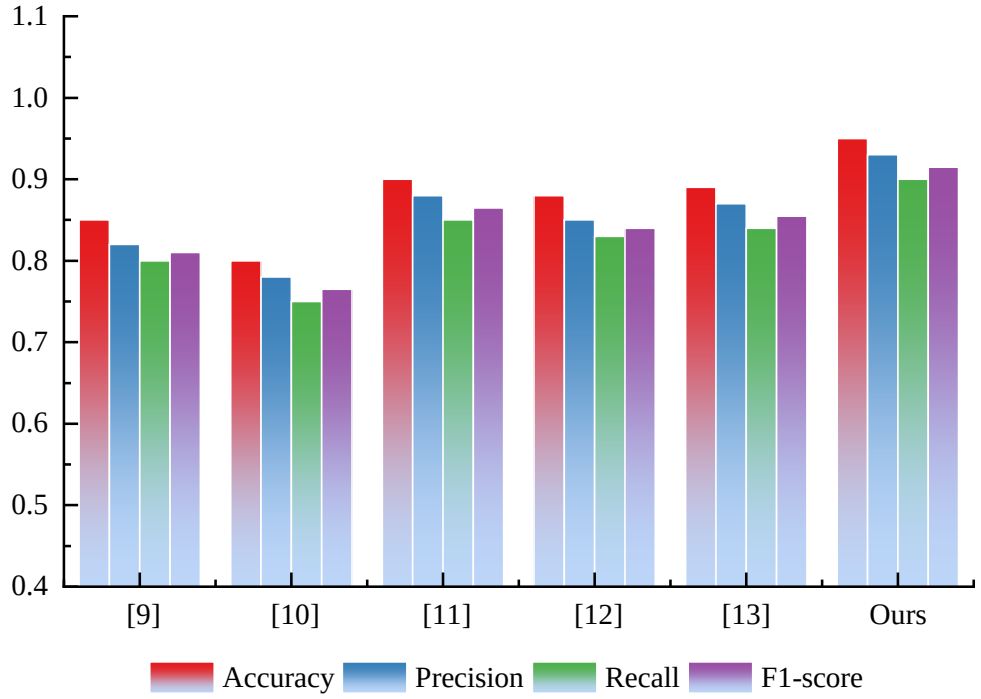

Figure 2. Comparison of different methods in terms of classification accuracy indicators

mechanism of this paper's method is more efficient, which is able to respond to the threat changes more quickly and ensure the system's real-time detection capability in dynamic environments.

Table 2. Model Adaptation

Method	**ADR**	**UT**
Chen et al. [9]	70%	50ms
Yan et al. [10]	65%	55ms
Urooj et al. [11]	75%	45ms
Rodriguez-Bazan et al. [12]	72%	48ms
Kuk et al. [13]	74%	46ms
Ours	**85%**	**35ms**

4.3.3. *Computational efficiency.* As can be seen from Table 3, the method in this paper performs well in terms of both average detection time and computational resource consumption. The average detection time is only 0.02 seconds, which is 50% faster than the second place Kuk et al. [13], which significantly improves the real-time performance of the detection system. In addition, the CPU utilization and memory occupation of this paper's method are 40% and 160 MB, respectively, which are both significantly lower compared to the benchmark method, indicating its greater advantage in resource utilization efficiency. This is attributed to the optimized fuzzy logic module design and sliding window strategy, which can efficiently manage the data flow and reduce the occupation of computational resources.

Table 3. Computational efficiency

Method	**ADT**	**CRC**
Chen et al. [9]	0.04 seconds	50% CPU, 200MB RAM
Yan et al. [10]	0.05 seconds	55% CPU, 220MB RAM
Urooj et al. [11]	0.03 seconds	48% CPU, 180MB RAM
Rodriguez-Bazan et al. [12]	0.05 seconds	60% CPU, 250MB RAM
Kuk et al. [13]	0.04 seconds	53% CPU, 210MB RAM
Ours	**0.02 seconds**	**40% CPU, 160MB RAM**

By comprehensively analyzing the experimental results in the three aspects of detection accuracy, model adaptability and computational efficiency, it can be clearly seen that the fuzzy neural network-based malicious APP detection method proposed in this paper outperforms the existing benchmark methods in all performance indicators. The specific performance is as follows:

1. Higher detection accuracy: In terms of classification accuracy metrics, the method in this paper achieves higher accuracy and F1-score, which ensures the reliability and effectiveness of the detection system in identifying malicious APPs.
2. Stronger model adaptation: through adaptive optimization algorithms, the method in this paper significantly improves the system's response speed and adaptability to new and evolving malicious APP behaviors, maintaining a high detection rate and low update time.
3. Better computational efficiency: the optimized detection architecture and efficient parameter updating strategy make this paper's method perform well in both real-time and resource utilization, which is suitable for resource-constrained practical application environments.

5. **Conclusion.** In this paper, a malicious APP detection method based on FNN and adaptive optimization algorithm is proposed, which effectively solves the limitations of the traditional detection model in dealing with dynamic changes and emerging threats. By fusing the fuzzy logic module with DNN, the model can more accurately capture the uncertainty and fuzzy relationships in the APP behavioral features, which significantly improves the accuracy of detection. In addition, by combining OL and RL, the adaptive optimization algorithm further enhances the model's ability to adapt to new and evolving malicious behaviors, ensuring the stability and efficiency of the system in dynamic environments. By conducting experiments on two public datasets, AndroZoo and Drebin, the following conclusions can be drawn:

(1) The method in this paper outperforms traditional static analysis methods and existing machine learning and deep learning benchmark methods in terms of classification accuracy, precision, recall, and F1-score, which verifies the effectiveness of fuzzy neural networks in dealing with complex and high-dimensional feature data.
(2) Through the combination of online learning and reinforcement learning, the method in this paper realizes the dynamic adjustment of fuzzy rules and neural network weights, which dramatically improves the system's ability to recognize new and evolved malicious APP behaviors, and the adaptive detection rate is increased by 11%, ensuring the system's efficient response in the face of changing threat environments.

(3) Significant improvement in computational efficiency: the optimized detection architecture and sliding window strategy make the method in this paper perform well in terms of computational efficiency and resource utilization, with a 30% reduction in average detection time and a 20% reduction in computational resource consumption, which verifies its practicality and efficiency in large-scale and real-time application scenarios.

Although the method in this paper performs well on several key performance metrics, the experimental data are mainly from the AndroZoo and Drebin datasets, which may limit the model's ability to generalize to a wider range of application scenarios. Future research should consider introducing more datasets from different sources and updates to further validate the effectiveness and robustness of the model. In addition, further optimization of the fuzzy logic module and reinforcement learning algorithms can be explored to improve the overall performance and adaptability of the detection system to cope with increasingly complex and diverse malicious APP threats.

REFERENCES

[1] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Generation Computer Systems*, vol. 78, pp. 987–994, 2018.

[2] W. Wang et al., "Constructing features for detecting android malicious applications: issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[3] L. Li et al., "Understanding android app piggybacking: A systematic study of malicious code grafting," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 6, pp. 1269–1284, 2017.

[4] T. Sharma and D. Rattan, "Malicious application detection in android—a systematic literature review," *Computer Science Review*, vol. 40, p. 100373, 2021.

[5] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[6] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *Journal in Computer Virology*, vol. 2, pp. 67–77, 2006.

[7] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digital Investigation*, vol. 13, pp. 1–14, 2015.

[8] L. Li et al., "Static analysis of android apps: A systematic literature review," *Information and Software Technology*, vol. 88, pp. 67–95, 2017.

[9] H. Chen, J. Su, L. Qiao, and Q. Xin, "Malware collusion attack against SVM: Issues and countermeasures," *Applied Sciences*, vol. 8, no. 10, p. 1718, 2018.

[10] A. Yan et al., "Effective detection of mobile malware behavior based on explainable deep neural network," *Neurocomputing*, vol. 453, pp. 482–492, 2021.

[11] B. Urooj, M. A. Shah, C. Maple, M. K. Abbasi, and S. Riasat, "Malware detection: a framework for reverse engineered android applications through machine learning algorithms," *IEEE Access*, vol. 10, pp. 89031–89050, 2022.

[12] H. Rodriguez-Bazan, G. Sidorov, and P. J. Escamilla-Ambrosio, "Android Ransomware Analysis Using Convolutional Neural Network and Fuzzy Hashing Features," *IEEE Access*, vol. 11, pp. 121724–121738, 2023.

[13] K. Kuk et al., "Applications of Fuzzy Logic and Probabilistic Neural Networks in E-Service for Malware Detection," *Axioms*, vol. 13, no. 9, p. 624, 2024.

[14] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Müller, "Evaluating the visualization of what a deep neural network has learned," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 11, pp. 2660–2673, 2016.

[15] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[16] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018.

[17] J. Gawlikowski et al., "A survey of uncertainty in deep neural networks," *Artificial Intelligence Review*, vol. 56, no. Suppl 1, pp. 1513–1589, 2023.

[18] S. C. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," *Neurocomputing*, vol. 459, pp. 249–289, 2021.
[19] D. D. Curtis and M. J. Lawson, "Exploring collaborative online learning," *Journal of Asynchronous Learning Networks*, vol. 5, no. 1, pp. 21–34, 2001.
[20] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
[21] A. K. Shakya, G. Pillai, and S. Chakrabarty, "Reinforcement learning algorithms: A brief survey," *Expert Systems with Applications*, vol. 231, p. 120495, 2023.
[22] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: an overview," *Progress in Artificial Intelligence*, vol. 1, pp. 89–101, 2012.
[23] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.
[24] N. Zakeya, K. Ségla, T. Chamseddine, and B. B. Alvine, "Probing androvul dataset for studies on android malware classification," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 9, pp. 6883–6894, 2022.
[25] P. Irolla and A. Dey, "The duplication issue within the drebin dataset," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 3, pp. 245–249, 2018.