electronics

Article

AIBot: A Novel Botnet Capable of Performing Distributed Artificial Intelligence Computing

Hao Zhao, Hui Shu *, Yuyao Huang and Ju Yang

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450000, China
* Correspondence: shuhui123@126.com

Abstract: As an infrastructure platform for launching large-scale cyber attacks, botnets are one of the biggest threats to cyberspace security today. With the development of network technology and changes in the network environment, network attack intelligence has become a trend, and botnet designers are also committed to developing more destructive intelligent botnets. The feasibility of implementing distributed intelligent computing based on botnet node resources is analyzed with regard to the aspects of program size, communication traffic and resource occupancy. AIBot, a botnet model that can perform intelligent computation in a distributed manner, is proposed from the attacker's perspective, which hierarchically deploys distributed neural network models in the botnet, thereby organizing nodes to collaboratively perform intelligent computation tasks. AIBot enables the distributed execution of intelligent computing tasks on a cluster of bot nodes by decomposing the computational load of a deep neural network model. A general algorithm for the distributed deployment of neural networks in AIBot is proposed, and the overall operational framework for AIBot is given. Two classical neural network models, CNN and RNN, are used as examples to illustrate specific schemes for deploying and running distributed intelligent computing in AIBot. Experimental scenarios were constructed to experimentally validate and briefly evaluate the performance of the two AIBot attack modes, and the overall efficiency of AIBot was evaluated in terms of execution time. This paper studies new forms of botnet attack techniques from a predictive perspective, aiming to increase defenders' understanding of potential botnet threats, in order to propose effective defense strategies and improve the botnet defense system.

Keywords: cybersecurity; botnet; CNN; RNN; distributed computing

Citation: Zhao, H.; Shu, H.; Huang, Y.; Yang, J. AIBot: A Novel Botnet Capable of Performing Distributed Artificial Intelligence Computing. *Electronics* **2022**, *11*, 3241. https://doi.org/10.3390/electronics11193241

Academic Editor: Fernando De la Prieta

Received: 9 September 2022
Accepted: 3 October 2022
Published: 9 October 2022

1. Introduction

The frequent occurrence of security incidents in cyberspace, with attacks such as sensitive information theft, phishing fraud, crypto-ransom threats, spam bombing, malware distribution and distributed denial of service, has caused widespread concern in academia and industry. Botnet, the infrastructure platform for launching these large-scale attacks, is considered one of the biggest security threats in cyberspace today [1]. A botnet is a general-purpose computing platform that can be remotely controlled by an attacker, built using many non-cooperative user terminals in the compromised cyberspace and usually consisting of a botmaster, a command and control (C&C) channel and bots [2]. As the network environment has changed, botnets have gone through different stages of development, from a single attack in the early days to a wide range of attacks today, and their scope of influence has expanded from PCs and servers to smart mobile terminals and IoT devices, with increasingly complex construction techniques, diversified performance forms, diversified attack methods and gradually increasing damage capabilities, seriously affecting network security and posing a huge challenge to defenders.

The rapid development of artificial intelligence technologies has had a significant impact on the current cyberspace security posture. On the one hand, defenders have achieved good results by using machine learning, deep learning and other big data analysis

tools for intrusion detection systems, malware classification and abnormal traffic identification; on the other hand, this also gives attackers the possibility of making network attacks intelligent, such as through in-depth analysis of collected user data with neural network models to mine valid information for further attack actions. SNAP_R [3] is the first known automated, end-to-end spear-phishing campaign generator that uses Twitter user data to train recurrent neural network models to generate high click-through-rate phishing tweets and clustering algorithms to identify high-value targets. This type of cyber-attack mode using deep learning and other artificial intelligence technologies to achieve automation will become a huge threat to cyberspace security if combined with botnets and used by a botmaster to launch large-scale intelligent cyber-attack activities. Therefore, it is very important to study the botnet mechanism and the botnet evolution law, analyze the feasibility of artificial intelligence technology for botnet attack activities, predict new botnet forms and attack patterns and improve the existing botnet defense system in order to improve emergency response capabilities against botnet attack incidents and guarantee cyberspace security.

1.1. Botnet Evolution

The development and application of network technologies are intended to promote social progress, and Internet developers are happy to open source the latest technical achievements for the general technical community to study and discuss. However, due to the profit motive, there are numerous cases of positive technologies being maliciously exploited by hackers, which is particularly evident in botnet attack campaigns. For example, the Mirai botnet has taken advantage of the booming Internet of Things to control vast numbers of end devices to launch DDoS attacks [4]. This has led to the proliferation of IoT botnets, which are expanding the size of the network by hacking into end devices through username and password guessing and firmware vulnerability scanning [5]. The development and application of Web 2.0 technology have given rise to the popularity of online media such as social networks but also provided a platform for botmasters to exploit. Up to 40% of accounts on Facebook, Twitter and other popular online social networks (OSNs) belong to botnets [6]. Botmasters use online social networks to build command and control channels for a variety of attack, abuse and control activities, such as spamming and political activism [7]. In recent years, researchers have continued to track botnet evolution trends; analyzed the application of anonymous communication techniques, such as Tor, in botnet command and control mechanisms [8,9]; and proposed corresponding preventive measures [10]. Innovations in network technology have brought about changes in the network environment, giving rise to new means for network attacks and the emergence of new botnets that are more destructive and stealthy. Therefore, botnet prediction has become one of the main research directions in related fields [11], with the aim of proposing new possible types of future botnets from the attacker's perspective and forward-looking defense methods.

1.2. Motivation and Reason

For botmasters, a shift in the botnet attack paradigm towards intelligence is necessary. The large amount of data (such as user information, text, images and voice) obtained from end devices needs to be computationally processed to unlock value. In the field of data analysis, deep learning algorithms and models are the current mainstream techniques, and a previous study [12] showed a large improvement in solving similar problems compared to previous approaches. Sending huge amounts of raw data back to the cloud server for further centralized intelligent analysis will undoubtedly generate huge amounts of network traffic, which places greater demands on the building of botnet back channels and also increases the risk of exposure. If the data can be intelligently computed and processed at the node side, and only the intermediate or final results after computation are returned, it will significantly improve the attack efficiency and reduce the scale of communication traffic. Therefore, running intelligent computational models right at the edge nodes to

compute and process the collected data is likely to be the evolving trend in botnet attack patterns. This is both a scenario for the application of artificial intelligence technology in botnet attack activities and also a reflection of the intelligence of botnet attacks.

It makes sense to deploy and run artificial intelligence algorithms, such as deep learning, in a botnet. Intelligent algorithms based on deep neural networks (DNNs) have made significant progress in recent years, mainly due to the dramatic increase in hardware computing power. In botnets, controlled clusters of scaled nodes are very substantial idle computing resources. The types of nodes in the botnet cover multiple layers of network devices, most of which have difficulty performing complex intelligent computing tasks independently. However, the neural network model is capable of distributed deployment after corresponding modification and decomposition of computational units; i.e., the computational tasks are performed jointly by multiple nodes. The botmaster can deploy trained neural network models into the botnet in a distributed manner and organize relevant nodes to collaborate in performing intelligent computing tasks, thereby completing attack tasks, such as data classification, prediction and generation; such an attack model is perfectly feasible in theory.

In summary, how distributed node resources can collaborate to accomplish complex intelligent computing tasks is the focus of botnet designers, and the key to the problem lies in studying and proposing distributed deployment strategies and methods for neural networks that fit with the botnet architecture. In fact, distributed artificial intelligence is a current research hotspot [13], but such research mainly focuses on parallel training of and optimization acceleration in neural network models in large-scale distributed environments, while botnet controllers focus on the distributed deployment of optimized algorithmic models to various nodes and then collaborative execution of intelligent computational tasks. To investigate this potential cybersecurity threat, this paper proposes AIBot, a botnet model that can be distributed to deploy deep neural network algorithms and that hierarchically deploys distributed neural network models in botnets, which in turn can organize nodes to collaboratively perform intelligent computing tasks.

1.3. Contributions and Structure

Based on the current cyberspace security situation, this paper analyzes the botnet evolution law and discusses the rationality and necessity of organizing botnet nodes to perform intelligent computing tasks in a distributed manner from the attacker's perspective. The difficulties in designing this type of botnet in terms of program size, communication traffic and resource consumption are comprehensively analyzed, possible solution ideas are discussed and a botnet model (AIBot) that can perform intelligent computing tasks is proposed on this basis. Unlike traditional botnets that send the collected sensitive data back to the cloud center for processing, AIBot deploys the neural network model to the end devices in a distributed manner, analyzing and processing data directly at the bot nodes, thus making full use of the computing resources of the botnet. A formal representation of the operation process of AIBot is given, its overall operational framework is given, the distributed deployment schemes of two neural network models (CNN and RNN) in the botnet are elaborated and the architecture is presented in a hierarchical manner. An experimental scenario is constructed to mimic the real network environment, and simple feasibility verification and performance evaluation of AIBot are conducted. The limitations of AIBot are analyzed, and potential defense strategies are proposed. Intelligent botnet attack activities will cause great harm to cyberspace security, and it is necessary to study their theoretical basis and implementation principles. Therefore, this paper proposes a new form of botnet attack from a predictive perspective, aiming to increase the understanding of the new botnet threat by conducting relevant research ahead of the attackers and thus suggesting effective defenses.

The remainder of this paper is structured as follows: Section 2 presents the research background and related work. Section 3 analyzes the difficulties of the work in this paper and the corresponding solution ideas. Section 4 describes the botnet model (AIBot)

proposed in this paper in terms of its formal representation, the distributed deployment of neural networks and architecture, etc. Section 5 presents experiments and a brief evaluation of the attack effectiveness and execution efficiency of AIBot. Section 6 discusses the limitations of AIBot and possible defensive measures. Section 7 provides a summary of the whole paper.

2. Background and Related Work

In this paper, we study a novel attack model combining a botnet and artificial intelligence techniques. This is a topic with limited directly related research, so we describe the background and related work in terms of botnet attacks, deep neural networks and distributed artificial intelligence. As shown in Figure 1, we first classify botnet attacks according to different attack locations and compare similarities and differences between the work in this paper and virtual currency mining; then, we introduce two typical deep neural networks and analyze their applications in network security; finally, we introduce the concept of distributed computing and the related work on distributed neural networks.

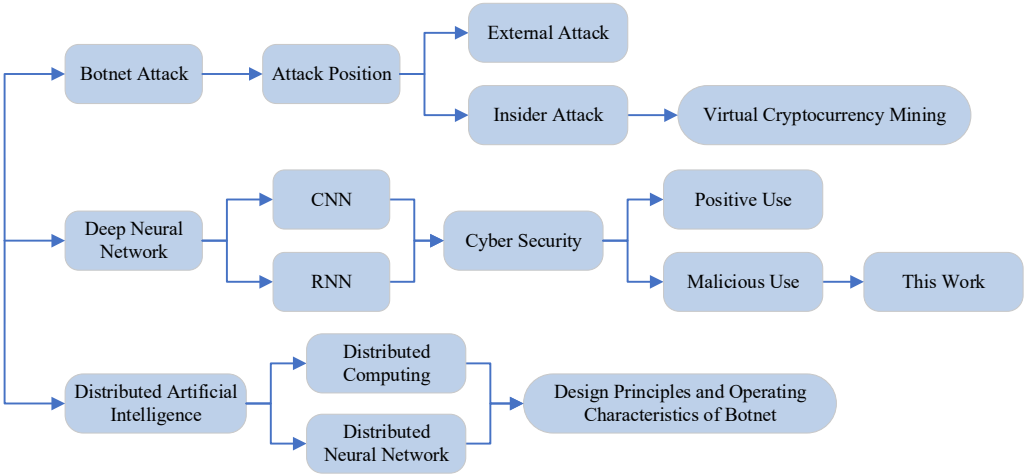

Figure 1. Background and related work.

2.1. Botnet Attack

The main purpose of an attacker when building and controlling a botnet is to control a large number of nodes to launch cyber-attack activities for financial or political gain. The types of attacks are classified from the perspective of the location of the attack: (i) external attacks: distributed denial of service attacks (DDoS), click fraud [14], phishing, spamming [15], scanning probes, etc.; and (ii) internal attacks: information harvesting, virtual currency mining [16], crypto-extortion [17], etc. In the former case, the attack targets are located outside the botnet, while in the latter case, the attack activity is carried out inside the botnet. Virtual currency mining refers to the control of a large number of bot node resources for "mining" in return for virtual cryptocurrency. This attack has similarities to this paper's research in that it exploits the computational resources of nodes that are idle for long periods to perform complex computing tasks. The difference is that "mining" has a mature "pool-miner" mechanism that can be directly transplanted to botnets, while there is no solution for deploying distributed neural network models in botnets. As cyber-attack awareness and defensive capabilities increase, traditional botnet attack campaigns are developing more mature defensive and mitigation strategies, so expanding internal attack methods is likely to be one of the evolving directions for botnets. However, the existing literature mainly focuses on the construction level of botnets and less on the evolution of botnet attack behavior.

2.2. Deep Neural Network

In recent years, artificial intelligence technology, as represented by deep learning, has seen rapid developments, and it is a hotspot for current research. Deep learning is mainly implemented based on deep neural networks, and the representative algorithms are convolutional neural networks (CNNs) and recurrent neural networks (RNNs). A CNN is a class of feedforward neural network (FNN) with convolutional computation and a deep structure; these networks have representation learning capabilities and are capable of shift-invariant classification of input information according to their hierarchical structure. Recurrent neural networks (RNNs) take sequential data as input and utilize recursion in the direction of sequence evolution with all nodes (recurrent units) connected in a chain, and long short-term memory networks (LSTMs) are the most widely used among them.

Deep learning algorithms are mainly used to solve problems such as image classification, speech recognition and natural language processing, and their performance has been greatly improved compared to traditional machine learning algorithms. Researchers have thus been inspired to apply deep neural network models to the field of cybersecurity, achieving good results in network traffic classification, malicious domain name detection and attack identification [18]. In the field of botnet countermeasures, existing research related to deep learning has focused on botnet detection and identification [19]. There are no publicly available cases of botnet developers applying intelligent computing models, such as deep neural networks, in their attack campaigns.

2.3. Distributed Artificial Intelligence

With the growth of the Internet, some applications have developed high requirements for computing power, which are time-consuming if traditional centralized computing is used. Distributed computing is used to solve exactly this type of complex problem by breaking down the computational task into many smaller parts and distributing them to multiple computers for processing. Compared with centralized computing, distributed computing can reduce overall computing time and greatly improve computing efficiency. As the network environment has changed, concepts such as fog computing [20] and edge computing [21] have been introduced. What these computing patterns have in common is the full utilization of the computing resources of the end device. A cluster of nodes controlled by a botnet can be seen as a distributed computing platform used to perform attack tasks, and this distributed architecture fits in with distributed computing systems. Therefore, the research in this paper also belongs to the category of distributed computing to some extent, but there are special features of botnet-based distributed AI computing that cannot be directly applied to the above-mentioned computing pattern.

There are currently two imperfect solutions for machine learning systems on end devices: (i) one is to aggregate data collected by edge nodes to the cloud for computation; (ii) the other is to execute simple machine learning models (e.g., SVM) independently on end devices. The former results in significant communication costs, while the latter is lacking in performance. With the exponential growth of training data and computation, distributed AI was born, with the main purpose of spreading the work of machine learning across multiple devices and transforming a centralized system into a distributed one. Current research in distributed AI focuses on the efficient parallelization of the training process and the establishment of consistent models, with the main aim of rationally distributing training data and accelerating the training process [22], which is not the same as the purpose of this paper. In terms of distributed computing of neural network models, Teerapittayanon [23] et al. proposed BranchNet, a neural network model with multiple exit points based on the idea that a shallow convolutional neural network would be able to correctly classify most of the samples, and designed a distributed neural network DDNN [24] on this basis. The above work has some reference value, but the research in this paper focuses on new attack modes of botnets and explores the distributed deployment and application of different types of neural network models in botnets under multiple scenarios, and it needs to

be considered in conjunction with the design principles and operational characteristics of botnets.

3. Difficulties and Solutions

3.1. Difficulties

The following is an analysis of the difficulties in designing this type of botnet from the perspective of botnet developers in terms of bot program size, communication interaction traffic and node resource consumption:

- **Bot program size**: In order for a device to run the corresponding intelligent learning model, a bot program needs to be loaded with the structure and parameters of the computational model, and neural networks often have thousands or even tens of thousands of parameters. Therefore, the size of this type of bot program is positively correlated with the number of parameters contained in the model and is significantly larger than that of traditional bot programs;
- **Communication interaction traffic**: Due to the large scale of the numbers of neural network parameters, the downloading of the files may generate extensive communication traffic. On the other hand, although multiple nodes working together to complete the computation process can take advantage of the number of nodes in the botnet, this also generates additional communication traffic for collaborative interactions between nodes as a result;
- **Node resource consumption**: When a botnet utilizes distributed node resources to collaboratively perform intelligent computing tasks, the CPU needs to perform a large number of parallel computations in a short time, and the computational resources of the nodes will be in a high occupancy state. Although most of the controlled nodes are in an unattended state, the long-term high occupancy rate still increases the risk of botnet exposure.

3.2. Solutions

Based on the analysis in the previous section, we argue that attackers should consider both the commonality of botnets, in terms of detection resistance and stealth, and the specificity of combining distributed deployment and intelligent computational models when building botnets that can perform AI computational tasks. As shown in Figure 2, the following potential solutions are proposed from the attacker's perspective for the difficulties that may be faced in building this type of botnet.

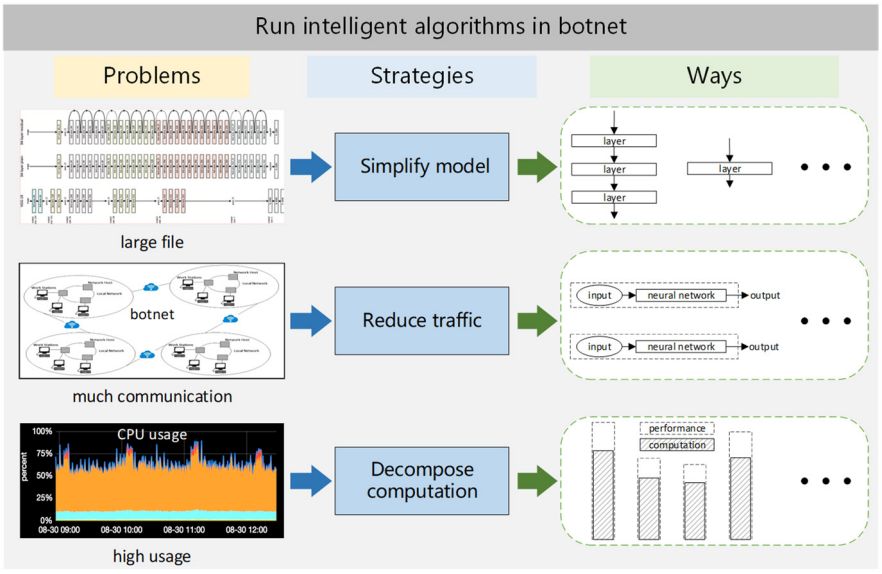

Figure 2. Idea analysis.

- **Simplify model**: In order to accommodate the characteristics of the botnet, the algorithmic model needs to be simplified. In fact, the performance of a fully trained shallow neural network model is basically able to meet the requirements when processing the task. The reason for increasing the number of neural network layers is to further improve the performance of the model, but at the cost of an exponential increase in the number of parameters. Considering the balance between program size and model performance, organizing nodes to run a simpler neural network collaboratively is a feasible solution. For CNNs, the number of model layers is scaled down to reduce the number of parameters while ensuring usability; for RNNs, the number of parameters in the recurrent units can be scaled down by setting the vectors to a lower dimensionality.
- **Reduce traffic**: Reducing the traffic interactions between nodes due to the computation process is an important part of making a botnet run intelligent algorithms. In order to reduce the large number of data interactions for intermediate results in the computation process, when allocating computation tasks, one can try to let a single node run the complete computation unit, complete the process of intermediate results inside the node and only output the final result of the computation unit externally. For the communication generated by the loaded model, the download traffic is reduced by the "simplify model" approach described above. For the input sources, the input data (e.g., images, corpus, etc.) are obtained locally at the node; i.e., the data collection and processing process is completed locally at the node.
- **Decompose computation:** The following methods can be used to decompose the computation quantity. First, the computation quantity of each node can be allocated according to the device performance—i.e., the hardware performance of the node can be determined by means of node sensing and detection, etc.—and the number of computation units in the node can be allocated according to a certain proportional coefficient so as to control the computation resource occupation rate of the node. Second, when the bot program executes the computation task, a waiting time can be added to the computation process (i.e., a time interval is inserted between the computation steps), and the duration of resource occupation can be actively reduced by means of intermittent execution.

4. Proposed Botnet Model

4.1. Formal Representation

The formal representation of the proposed botnet model that can run artificial intelligence algorithms is given below, and the composition of the botnet can be represented by Equation (1):

$$AIBot = \{Master,\ C\&C,\ ZOMBIE\} \tag{1}$$

Master is the botnet controller, and *C&C* represents the command and control channel of the botnet. *ZOMBIE* represents the set of bot nodes, which can be expressed by Equation (2), where n represents the size of the botnet:

$$ZOMBIE = \{zombie_i |\ i = 1,\ 2,\ \ldots\ n\} \tag{2}$$

Each bot node *zombie* can be seen as consisting of a *bot* and *host* together, *state* denotes the state of the node and *data* denotes the data collected by the current node (i.e., the input to the intelligent computation model), which can be expressed as Equation (3):

$$zombie_i = (bot_i,\ host_i,\ data_i,\ state_i) \tag{3}$$

If we denote the AI model deployed and run in the botnet as *Model*, then *Model* can be divided into multiple computational units, denoted as the sub-model *model*, and m denotes the number of units that can be divided, as in Equation (4):

$$Model = \{model_i |\ i = 1,\ 2,\ \ldots\ m\} \tag{4}$$

In AIBot, each node is assigned the corresponding computational task according to its computational performance, and the computational quantity is denoted by C. The quantitative performance index of the bot host is denoted by P. Then, the relationship between the computational unit *model* loaded in the *bot* program and *Model* can be roughly expressed with Equation (5), where r is the proportionality constant:

$$C\left(model_{bot_i}\right) = P(host_i) * C(Model) * r \tag{5}$$

The intelligent computational process in AIBot can be expressed with Equation (6), where the output of each computational unit is noted as *output*:

$$\left(data_i \rightarrow bot_i \rightarrow model_{bot_i}\right) \Rightarrow output_i \tag{6}$$

The final result (denoted as *Result*) is obtained by integrating the *output*s of the computational units of each node, and the corresponding computational strategy is adopted according to the type of neural network: for a CNN, the confidence level of the current output is first judged, and if it is higher than the set value, it is directly used as *Result*, and if it is lower than the set value, it is submitted to the upper layer network for further processing. For an RNN, each *output* is part of the final result, so the *output* is directly spliced to get the *Result*. Equation (7) gives a simple formal representation of the above process, and the specific details are discussed in Section 4.3:

$$Model_{Botnet} \Rightarrow Result = \begin{cases} \begin{cases} \text{directly } output, \text{ if confident} \\ \text{further calculations, if not confident} \end{cases}, & \text{(CNN)} \\ output_1 + output_2 + output_3 + \cdots, & \text{(RNN)} \end{cases} \tag{7}$$

4.2. Framework of AIBot

Before presenting the overall operational framework of AIBot, definitions and descriptions of the important processes in it are given.

Definition 1 (neural network decomposition). *The neural network model is decomposed using a single layer neural network that can be executed independently as a computational unit and, thus, deployed in a distributed manner. The formal division of the neural network model is given in Equation (4) in Section 4.1, and the specific decomposition strategy is specified here. The deep neural network model can be regarded as composed of the serialized connections of each single-layer neural network. The original model is denoted as Model, the sub-model is denoted as model, and each layer network can be represented as l_1. Taking the division into three sub-models as an example, each sub-model contains several network layers, and the decomposition of the neural network model can be further expressed as Equation (8):*

$$Model \Leftrightarrow \begin{bmatrix} model_1 \\ \downarrow \\ model_2 \\ \downarrow \\ model_3 \end{bmatrix} \rightleftharpoons \begin{bmatrix} (l_1) \\ \downarrow \\ (l_2 \rightarrow l_3) \\ \downarrow \\ (l_4 \rightarrow l_5 \rightarrow l_6) \end{bmatrix} \tag{8}$$

Definition 2 (node state identification). *Based on the node's hardware performance H (h1, h2, ...), system configuration S (s1, s2, ...) and network environment N (n1, n2, ...), the state of the node is identified comprehensively so as to determine the role positioning of the node and the quantity of computation undertaken by the node in the subsequent attack tasks. Equation (9) gives the quantitative calculation method for node state identification, where α, β, and χ denote the weighting parameters of each influencing factor, respectively. AIBot gets the node state at the node side and feeds it to the server.*

$$state(n) = \alpha H + \beta S + \chi N = \sum_i \alpha_i h_i + \sum_i \beta_i s_i + \sum_i \chi_i n_i \tag{9}$$

Definition 3 (adaptive network structure). *The organization of multiple nodes for intelligent computing requires the support of different network structures, such as "parallel" and "tandem" structures. Denoting the node as n, the connection relationship between any two points can be expressed as Equation (10), then the different network structures can be expressed as Equations (11) and (12), A and B represent the multi-node "parallel" and "tandem", respectively. AIBot achieves adaptive transformation of the network structure by changing the connection relationship between nodes.*

$$link(n_i, n_j) = \begin{cases} 1, & connected \\ 0, & unconnected \end{cases}, \ i \neq j \tag{10}$$

$$network_A = \left\{ \left[link(n_i, n_j) = 1, j = m \right] \cup \left[link(n_i, n_j) = 0, j \neq m \right] \right\} \tag{11}$$

$$network_B = \left\{ \left[link(n_i, n_j) = 1, j = i + 1 \right] \cup \left[link(n_i, n_j) = 0, j \neq i + 1 \right] \right\} \tag{12}$$

Combining the definitions and descriptions of the main processes described above, Algorithm 1 gives the algorithm for distributed deployment of neural networks in AIBot:

Algorithm 1 *Distributed deployment of neural networks in AIBot*

Input: node set *ZOMBIE*, neural network model *Model*, initial network structure *network*
Output: *AIBot* that can perform intelligent computation in a distributed manner

```
n ← Size(ZOMBIE);        // Get the scale of zombie nodes
Recognition(ZOMBIE);     // Node state identification
FOR i = 1 → n DO         // Sorting by node state value
    FOR j = 1 → n DO
        IF (zombie[j].state > zombie[j + 1].state) THEN
            swap(zombie[j], zombie[j + 1]);
        END IF
    END FOR
END FOR
{model [1 → n]} ← Decompose(Model.layer, n)        // Neural network decomposition
FOR i = 1 → n DO         // Sorting by sub-model computation
    FOR j = 1 → n DO
        IF (C(model[j] > C(model[j + 1])) THEN
            swap(model[j], model[j + 1]);
        END IF
    END FOR
END FOR
FOR i = 1 → n DO
    Mapping(zombie[i], model[i]);        // Deploy sub-models to the corresponding zombie
END FOR
network' ← Reorganize(Model.type, network);        // Adaptive network structure
```

Figure 3 illustrates the overall operation of AIBot, and the steps are as follows:

I. The botmaster controls a large number of network devices as node resources for performing large-scale distributed computing;
II. The nodes are classified by obtaining various attributes through the **Node state identification** mechanism;
III. The botmaster launches the attack campaign and determines the type of neural network to be run based on the type of task;
IV. Neural network models are decomposed into distributable deployable form by applying **neural network decomposition** mechanisms according to the type of neural network;
V. **The adaptive network structure** mechanism is used to transform the interconnection relationship between nodes according to the neural network type;

VI. The distributed deployment of neural networks in the botnets is completed based on steps II and IV;
VII. Distributed intelligent computing tasks based on neural network models are performed on the basis of steps V and VI.

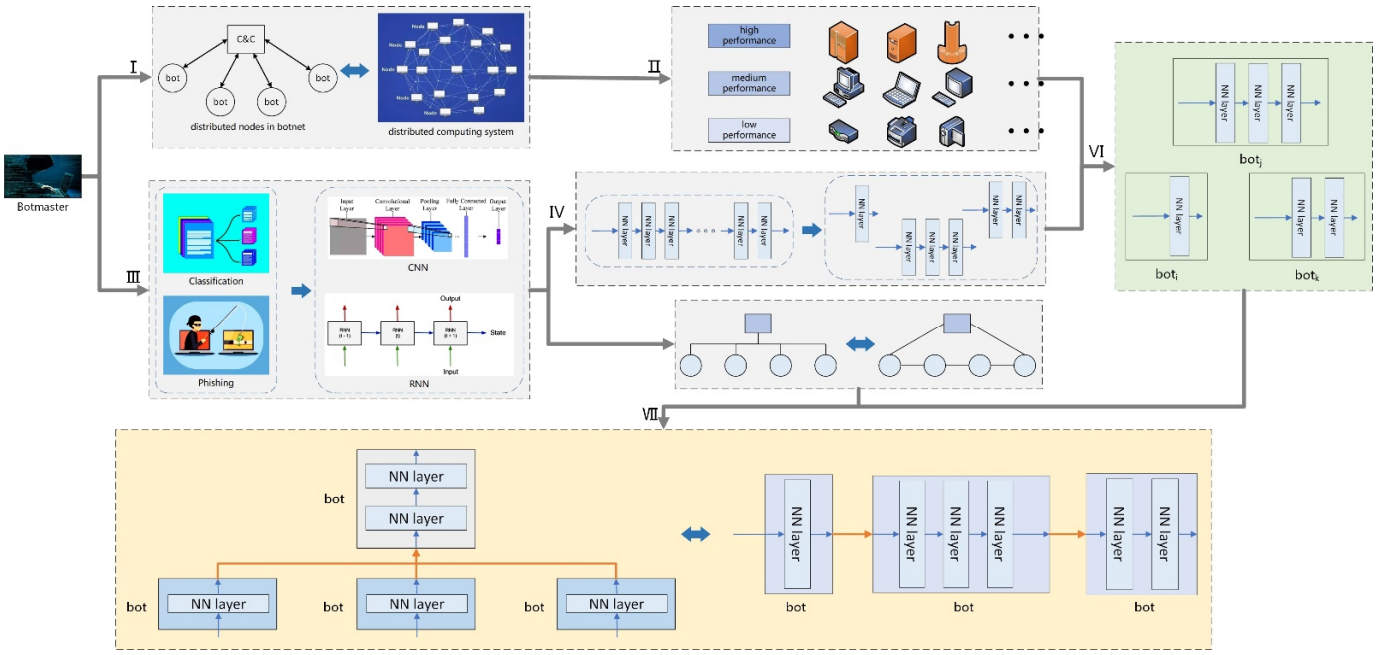

Figure 3. AIBot operational framework.

4.3. Two Implementations of AIBot

The AIBot botnet can theoretically meet the execution requirements of multiple intelligent computational models through automated computational quantity allocation and adaptive transformation of network structure. Although AIBot is a generalized model, there are still some implementation-level differences in performing different types of neural network computations. In this paper, we take a CNN and an RNN as examples to study the distributed deployment and operation of neural network models in AIBot. For convenience, the two bot models are abbreviated as C-AIBot and R-AIBot below, denoting the deployment and application of a CNN and an RNN in AIBot, respectively.

4.3.1. C-AIBot

It has been found that the convolutional neural network VGGNet (19 layers) was only about 4% more accurate (from 89% to 93%) than AlexNet (8 layers) when performing feed-forward inference calculations under the same conditions but with a 20-fold increase in running time and power consumption [25]. This disproportionate increase between resource usage and prediction accuracy is also evident for ResNet (152 layers); i.e., increasing the number of layers of the neural network up to a certain level results in a very small improvement in accuracy, which is not proportional to the operating cost. The above observations show that shallow neural networks already perform quite well in handling classification tasks, and the gap with deep neural networks is not too large. Based on the above facts, the central idea of designing a distributed deployment model of CNN in AIBot is to map the convolutional neural network hierarchically to the physical nodes of the botnet and to build a hierarchical distributed computing network according to the device type and performance configuration of the nodes, so as to achieve the purpose of performing artificial intelligence computing tasks.

In order to adapt the architecture for distributed deployment, the structure of the convolutional neural network needs to be adjusted and optimized first. The structure of a

convolutional neural network can be viewed as a linear sequence of multiple convolutional layers and fully connected layers. Given that the shallow network can also obtain good inference results in most cases, multiple branches are considered to be added to the original structure of the neural network, and the computational results of the early stages are used as the outputs of the corresponding branches. Specifically, the structure of the convolutional neural network is modified as follows: the shallow part of the neural network is reused to derive multiple branches of the neural network, thus adapting to bot nodes with varying hardware performance. As shown in Figure 4 (which simplifies the neural network layers except the convolutional layer), three branches are added to the original neural network and the outputs are expanded from one to four, forming four neural network models (branches 1~4) with increasing computational power, which can be deployed on four terminal nodes with low to high performance, respectively.

$$Confidence(y) = \sum_{l \in L} y_l \log y_l \tag{13}$$

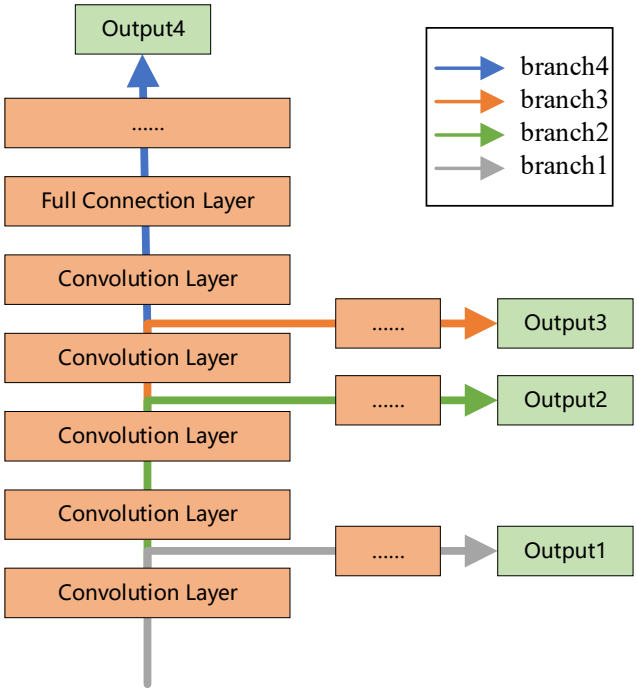

Figure 4. CNN with branches.

Shallow neural networks are usually less accurate in prediction than deeper ones, and the output of early branches is not always reliable. Therefore, this paper introduces the confidence index (denoted as *Confidence*) to measure the credibility of the output, using SoftMax as the regression function and borrowing the calculation method for the loss function to give the definition of *Confidence*, as in Equation (13) where y denotes the probability vector of all possible labels after SoftMax calculation and L is the set of all labels. To determine the confidence level of the results, the corresponding *Confidence* of each output is calculated and compared with the threshold *Th* obtained by prior training: if the classification result is credible, the neural network is withdrawn from the computation in that branch; if the classification result is not credible, it is submitted to the upper neural network layer to continue the computation.

The distributed deployment scheme of the CNN in AIBot after completing the adaptation modification is given below. The distributed nodes are first hierarchically divided into edge nodes, intermediate nodes and cloud nodes according to the device properties, and then the trained neural network models are mapped to the corresponding physical devices. From the edge to the cloud, the computing capability of the device is sequentially enhanced

to support deeper neural network operations. As shown in Figure 5, nodes at different levels divide the network into three layers, with the output of the neural network existing at each layer, and the confidence level of the classification result is judged to determine whether to submit the upper layer of the network for further operations.

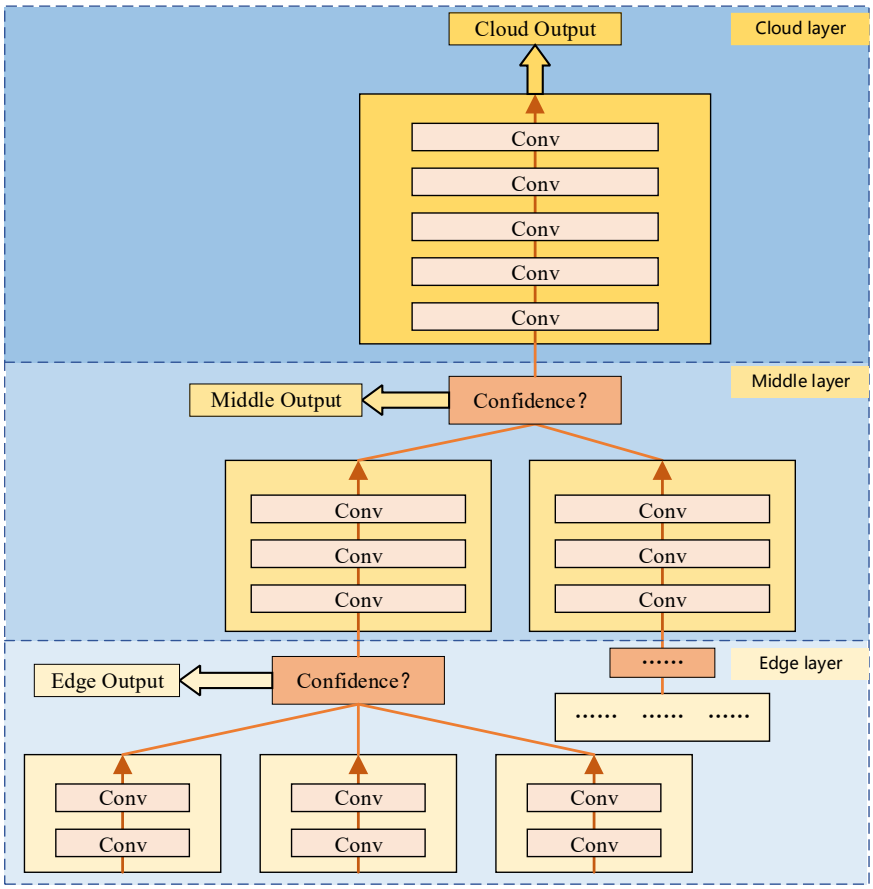

Figure 5. Distributed deployment of CNN in AIBot.

4.3.2. R-AIBot

An RNN takes sequence data as input and recurses in the direction of sequence evolution, and all recurrent units are connected in a chain-like manner. LSTM [26], as a type of recurrent neural network, solves the long-term dependency problem in general RNN networks through a conveyor belt mechanism and performs well in natural language processing problems, such as text generation. As shown in Figure 6, a recurrent neural network can be regarded as a chain computational network with each identical computational unit cyclically connected in series, and the output of the previous computational unit serves as the input of the next computational unit. In this paper, the repetitive computation units in RNN are represented as cells, and the parameters in the cells are uniquely determined after the neural network has completed being trained. Another major feature of RNN is sequential input and output: the first $n-1$ data points are sequentially serialized input and, after performing forward computation, the nth data point is inferred from the output result. The above analysis shows that an RNN cannot perform distributed parallel computation in the traditional sense but, based on the characteristics of its recurrent computation structure, it can distribute the overall computation quantity to multiple nodes and reduce the computation quantities of individual nodes, thus adapting to the needs of botnets.

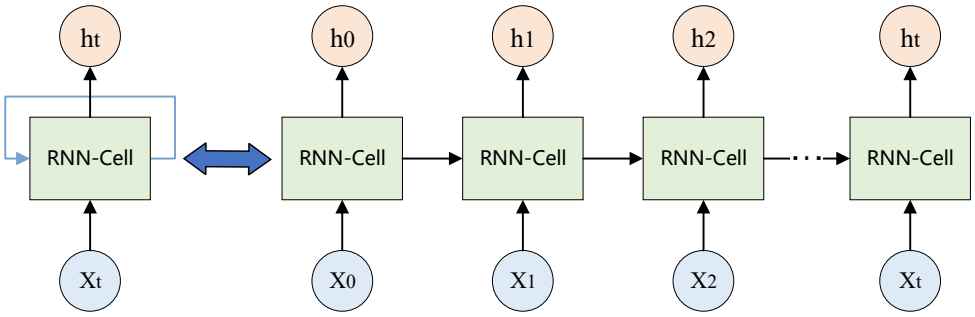

Figure 6. Chain structure of an RNN.

For distributed deployment of RNNs, the central idea is to distribute the computation according to the performance of the nodes; i.e., high-performance nodes take up more cycles of computation and low-performance nodes take up fewer cycles of computation. Figure 7 illustrates the distributed deployment scheme of an RNN with three nodes as an example: the nodes are written as i, j and k in order, and the operations are performed from node i, nodes j and k perform subsequent operations in turn and each node performs m_i, m_j and m_k cycles of computation respectively; the node performance is written as P. Then, the relationship between the node performance and the quantity of computation (in terms of the number of cycles) satisfies Equation (14):

$$\frac{P(i)}{m_i} = \frac{P(j)}{m_j} = \frac{P(k)}{m_k} \tag{14}$$

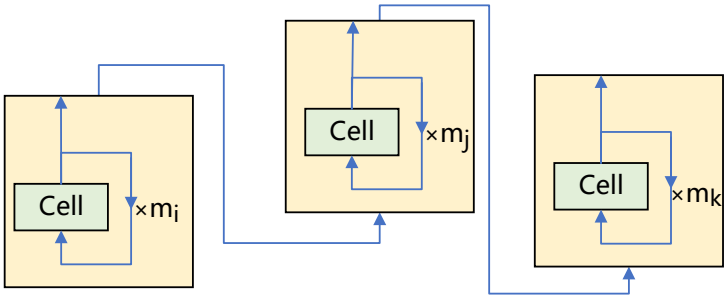

Figure 7. Distributed deployment of an RNN in AIBot.

Compared with CNNs, the decomposition and deployment of tRNN neural networks are more concise. However, the special serialized input and one-by-one generated output of RNNs need to be discussed and analyzed in the context of the distributed environment characteristics of botnets. When a trained RNN model performs computational tasks such as text generation, it needs an initial input, called the *seed*. The *seed* in AIBot is obtained locally at the node, and this approach is taken based on two main considerations: firstly, the seed text is obtained from the node where the target user is located, so that the generated text is more targeted; secondly, the computation task is started locally, which is in line with the principle of reducing the communication traffic between nodes. As shown in Figure 8, the controller in the cloud selects the target node and sends an attack command; *node* 1 receives the command, obtains the seed text locally and performs intelligent computation to generate *output*1; then, *nodes* 2, 3 and 4 generate *output*2, *output*3 and *output*4 according to the input data sent by the previous node in turn; finally, the generated complete text is returned to the controller. Each node determines the quantity of computation to be undertaken based on its own computational performance and, therefore, the length of the generated outputs varies.

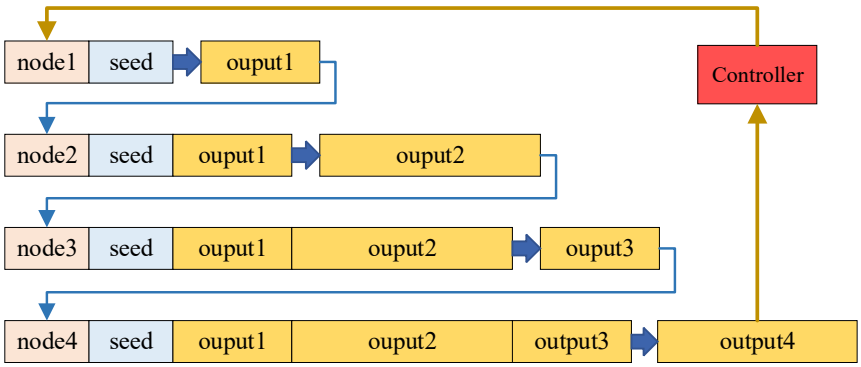

Figure 8. Task execution process.

4.3.3. Bot Design

Combining the above analyses, Algorithm 2 gives the overall design of the bot program in AIBot. AIBot in both modes is identical in terms of the overall operation flow but differs in the specific execution level due to the different types of neural network models.

Algorithm 2 *AIBot-bot*

```
1.  n_state ← Identify( );        // Get node state information
2.  Send(n_state, Server)         // Return node state to Server
3.  WHILE(TRUE) DO
4.      cmd ← Listen( );          // Listen for incoming attack commands
5.      IF(cmd.type = C) THEN     // Prepare to perform C-AIBot
6.          model_C ← download(cmd.file);
7.          Connect(cmd.link);            // Change the connection with other nodes
8.          output ← Calculate(input, model_C);     // Perform computing tasks
9.          IF(Confidence(output) < Th) THEN
10.             Send(output, Server);     // Return the results to the server
11.         ELSE THEN
12.             Send(output', upper_node);
13.         END IF
14.     ELSE IF(cmd.type = R) THEN    // Prepare to perform R-AIBot
15.         model_R ← download(cmd.file);
16.         Connect(cmd.link);
17.         output ← [input + Calculate(input, model_R)];
18.         IF(Length(output) < L) THEN
19.             Send(output, next_node);
20.         ELSE THEN
21.             Send(output, Server);
22.         END IF
23.     END IF
24. END WHILE
```

4.4. Architecture of AIBot

AIBot adopts a hybrid layered architecture that is divided into a cloud controlling layer, an intermediate processing layer and an edge computing layer from top to bottom according to the properties of the bot nodes and the network environment characteristics, which take up the corresponding management and computing tasks, respectively. As shown in Figure 9, the functional layers in the AIBot architecture are as follows:

- **Cloud controlling layer**: This layer is where the botmaster is located and has the highest authority over the control and management of the entire botnet, communicating with the intermediate processing layer through covert means, such as anonymous networks and blockchain protocols, to give orders and receive feedback. The botmaster is responsible for the state monitoring and daily maintenance of the botnet, releasing

version updates and plug-in tools and specifying attack targets and methods. In AIBot, the botmaster adjusts the intelligent computing model according to the actual effect and updates the relevant parameters and thresholds;

- **Intermediate processing layer**: This layer mainly consists of high-performance nodes in the botnet, including server nodes with high credibility and reliability, and adopts a backup redundancy mechanism to avoid "single point of failure". The intermediate processing layer acts as the equivalent of a C&C server in a traditional botnet, connecting directly with the underlying botnet nodes to receive commands from the higher-level controllers and to direct the lower-level edge nodes to perform specific tasks. In AIBot, the intermediate processing layer takes up part of the computational tasks of the deep neural network with an advantage in hardware performance, thus improving the accuracy of the computational results. Since AIBot needs to run multiple intelligent computational models, the intermediate processing layer is responsible for making the lower nodes adjust to the corresponding network structure;
- **Edge computing layer**: This layer concentrates most of the nodes in the botnet, including low-profile PCs, home routers and various lightweight connected endpoints, such as IoT devices. The inherent security flaws of such devices make them vulnerable to be captured as bot nodes, which, in turn, can serve as massively distributed computing resources under the control of botnets. The nodes in the edge computing layer can form different subnets of the botnet according to their network location to perform intelligent computing tasks, and the reachable public nodes are responsible for the communication with the upper layer network. As the target task and computational model change, the network structure of the edge computing layer is adjusted accordingly under the control of the intermediate processing layer.

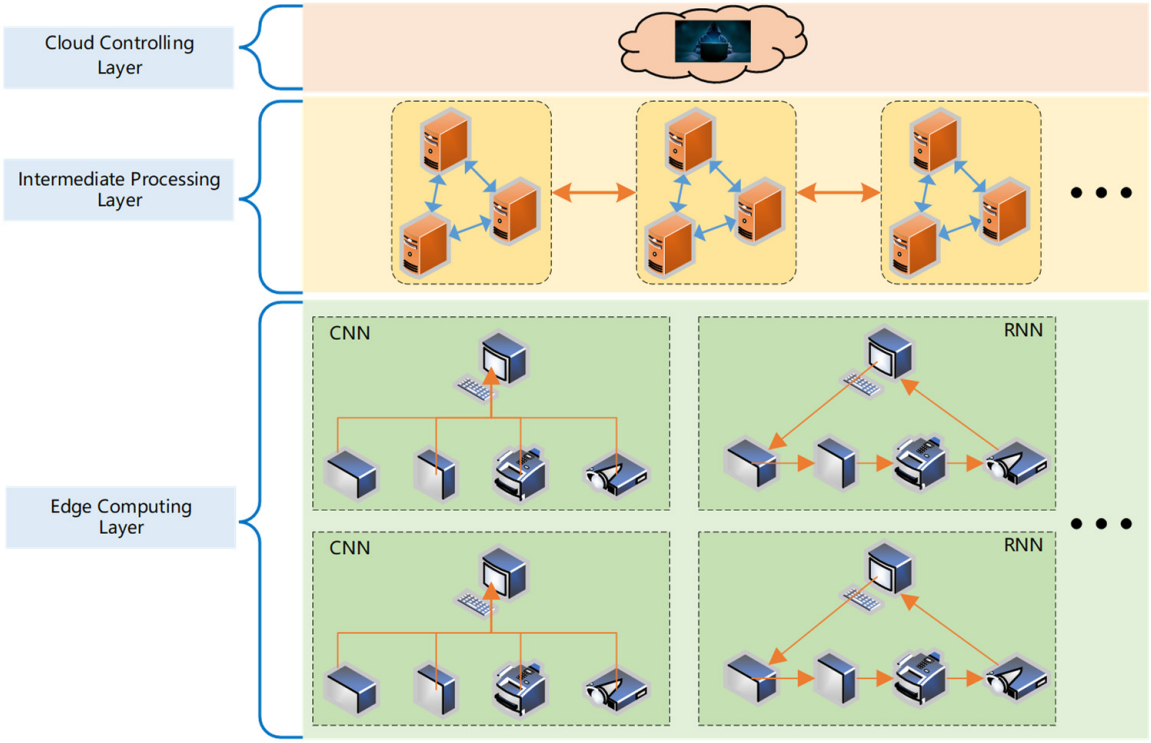

Figure 9. AIBot architecture.

5. Experiments and Evaluation

In this paper, we study the distributed deployment of two neural network models in botnets, the CNN and RNN; propose corresponding strategies and schemes; and perform experimental validation and performance evaluation, respectively. After the neural network

model is constructed, it needs to be trained and optimized with a large number of samples before it can be used for forward inference, and the training of the model includes complex computational processes, such as back propagation. In contrast, this paper focuses on the deployment and execution of the neural network model in the botnet, so the parameters of the model were determined by offline (joint) training, and then the trained model was deployed in the experimental network according to the scheme in Section 4.2. At the same time, the effectiveness of AIBot as a botnet model had to be evaluated in terms of its efficiency in performing attack tasks and other aspects. We simulated real nodes through Docker instances using the Ubuntu desktop system with version 18.04.5 and Docker 20.10.7. The base image loaded was the official Ubuntu image (latest), and the image with the embedded bot program was created and packaged for unified deployment.

5.1. C-AIBot Evaluation

5.1.1. Experimental Setup

Botnets obtain key information, such as user attributes, physical environment and social relationships, through comprehensive identification and analysis of data, such as pictures collected or documents and emails stored in similar nodes; e.g., comprehensive determination of target network attributes and types based on document data from multiple nodes. The experimental network was constructed as shown in Figure 10, consisting of six edge nodes and one cloud node in a distributed network architecture. The neural network deployed on the edge node contained only one convolutional layer, and the cloud node added two convolutional layers on top of reusing that convolutional layer. The hyperparameters of the CNN used in the experiments were as follows: the convolutional layer kernel size was 3×3, with stride of 1 and the padding of 1, and the pooling layer kernel size was 3×3, with stride of 1 and padding of 1. The experimental dataset was the classical CIFAR-10 [27], which contains 10 categories of image samples. The input of C-AIBot was a set of samples of the same category, and only the unique correct output existed after inference by the distributed neural network. Each output was essentially a probability vector used for classification, so we obtained the common output of the edge nodes by taking the maximum value of the corresponding position element. If it was to be handed over to the cloud for further processing, the output of the convolutional layer of each edge node was stitched together as the initial input of the neural network in the cloud, and the computational power of the cloud was used to maximize the use of sample information, thus improving the classification accuracy.

5.1.2. Performance Evaluation

Samples were exited from the calculation at the edge and in the cloud in a certain proportion, and accuracy refers to the proportion of all samples that were correctly classified, as given by Equation (15). Equation (13) in Section 4.3.1 gives the confidence calculation method, which determines whether the sample exited the calculation at the edge layer from the result of the comparison between the confidence *Confidence* and the threshold *Th*. The edge output rate—i.e., the proportion of samples that dropped out of the calculation right at the edge layer, is given by Equation (16). As shown in Figure 11, when increasing the value of *Th* in turn, the proportion of samples that drop out of the calculation at the edge gradually increases, while the overall accuracy changes more smoothly, reaching a maximum in the range of 7–8 for *Th*.

$$Accuracy_{C-AIBot} = \frac{Accuracy_{edge} * Sample_{edge} + Accuracy_{cloud} * Sample_{cloud}}{Sample_{edge} + Sample_{cloud}} \quad (15)$$

$$Edge\ Output\ rate = \frac{Sample_{edge}}{Sample_{edge} + Sample_{cloud}} \quad (16)$$

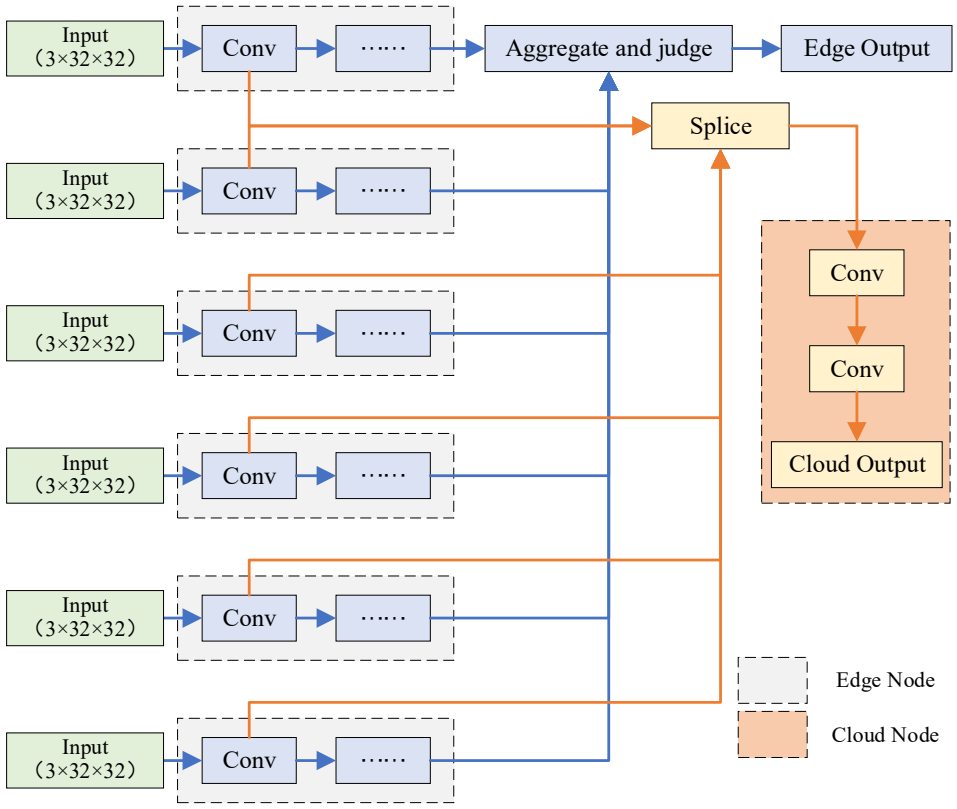

Figure 10. C-AIBot experimental network.

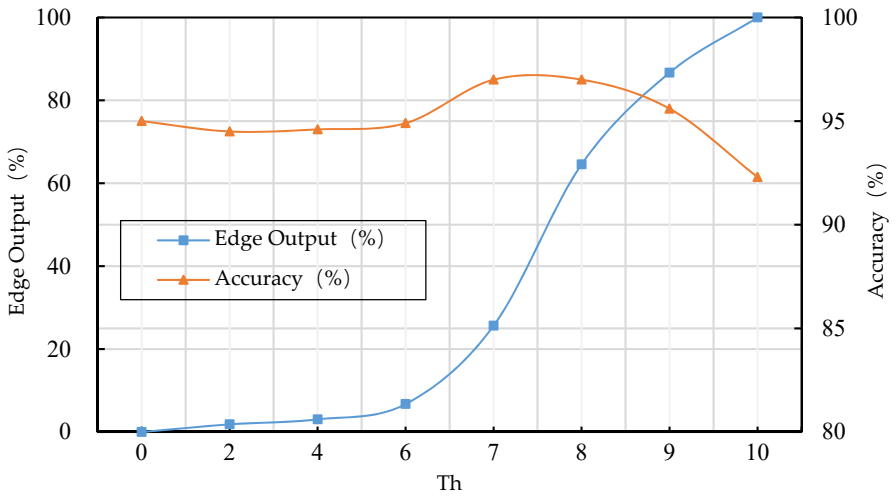

Figure 11. Impact of Th.

The communication traffic in C-AIBot consists of two main components: the traffic generated by sending inference results from the edge nodes to the sink, and the traffic generated by the edge nodes by uploading the output of the underlying neural network to the cloud. Figure 12 gives the variation in communication traffic (in bytes) and accuracy with the *Edge Output* rate. It can be seen that the communication traffic gradually decreased as the *Edge Output* rate increased, which is consistent with the fact that a large number of samples dropped out of the computation right at the edge. The communication traffic in the network was minimized when the *Edge Output* rate reached 100%, but the accuracy rate was relatively low at this point. Overall, the *Edge Output* rate was within a reasonable range of traffic and accuracy at between 60% and 80%.

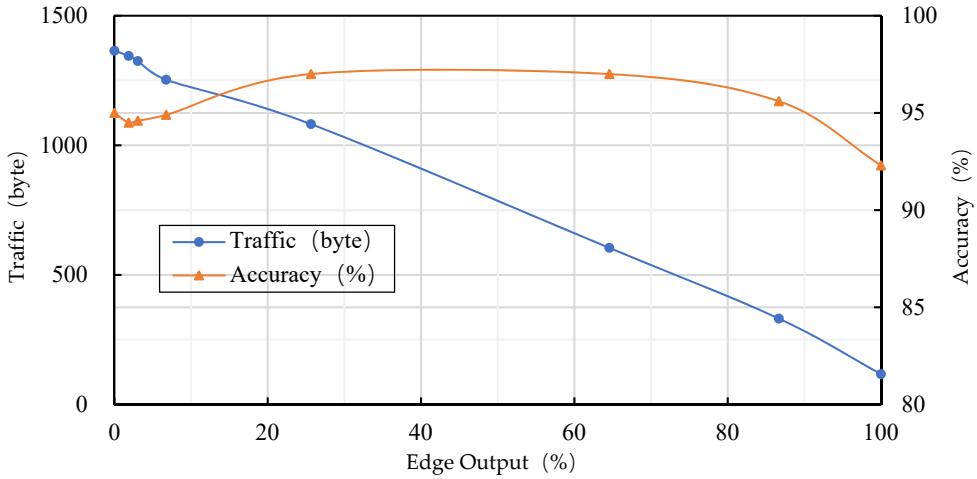

Figure 12. Variation in traffic and accuracy with edge output rate.

The impact of the number of edge nodes on the overall accuracy is an important aspect of evaluating C-AIBot. The state of individual nodes in a real botnet is not stable, so the random removal of edge nodes was chosen to simulate the real scenario. As shown in Figure 13, the accuracy decreased gradually as the edge nodes were removed sequentially, but the decreasing trend was not obvious in the first period, and the accuracy was still maintained at about 90% when two nodes were removed. The accuracy only started to show a significant downward trend when the number of nodes removed reached half. This indicates that the loss of a few nodes has a limited impact on the overall accuracy as long as most of the nodes are working properly.

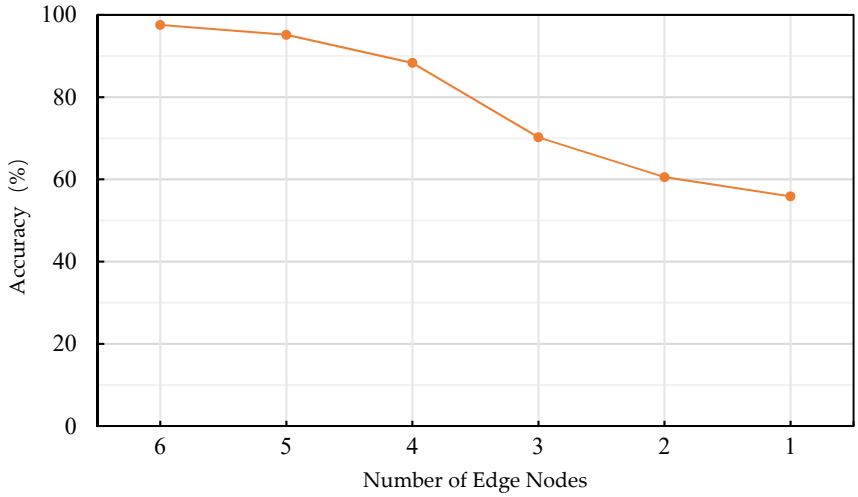

Figure 13. Variation in accuracy with the number of edge nodes.

5.2. R-AIBot Evaluation

5.2.1. Experimental Setup

Phishing attacks are one of the main forms of botnet attacks. Spear-phishing attacks, on the other hand, are more targeted towards launching phishing attacks at specific targets. R-AIBot precisely exploits the advantages of RNNs in text generation for distributed automated spear phishing. First, the cloud node selects the attack target, and then it obtains the seed data locally at the target, followed by each node executing the intelligent computing text-generation task one after another. The experimental network was constructed as shown in Figure 14, consisting of six edge nodes and one cloud node in a distributed network architecture. The LSTM text generation model was deployed on each node, and the experimental dataset came from a publicly available corpus of tweets on the Internet [28]

containing tens of thousands of real tweets. The LSTM model requires serialized inputs and outputs; i.e., subsequent text generation requires generated text as input. Therefore, all output data from the previous node were sent to the next node until a complete tweet was generated at the final node. In order to simulate the situation of nodes using local data for model training online, this paper used different subsets of the dataset as training sets to train multiple LSTM models, respectively, and deploy them to the corresponding edge nodes. The LSTM in the experiments was set up with a segment length of 20 and a state vector dimension of 32. The experiments used a specified amount of computation for each node to simulate the performance difference among nodes in the real network, and the cloud nodes were allowed to complete the computation process of text generation independently as a comparison.

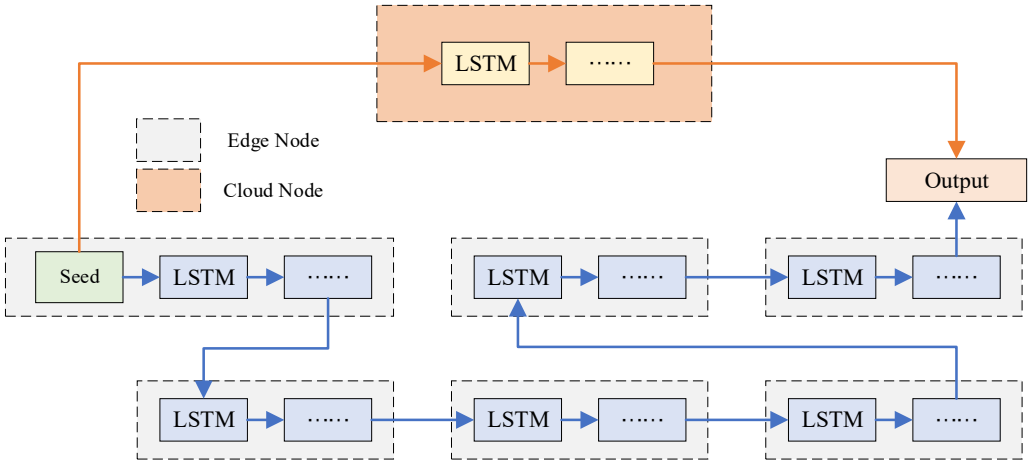

Figure 14. R-AIBot experimental network.

5.2.2. Performance Evaluation

The models deployed on each node were trained based on different subsets of data, so their validity had to be tested and verified first. A random sample of 20% from each data subset formed the test set for each node and the obtained results were compared with the accuracy of each node with their respective training sets. As shown in Figure 15, the LSTM models deployed on each node achieved high accuracy with their respective training sets, while their performances with the test set all declined but still managed to stay above 80%. In fact, methods such as increasing the number of neural network layers and adding a bidirectional LSTM can improve the performance of text generation models, but this is contrary to the principle of reducing the amount of node computation.

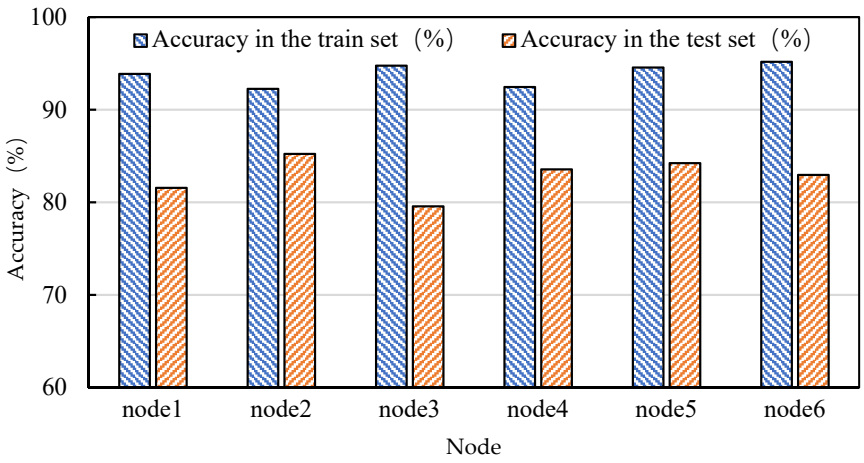

Figure 15. Accuracy of each node in the training set and test set.

In the experimental validation of the multi-node text generation, the ratio of computation assigned to each node was 1:1:2:2:3:3 in order, while the LSTM model (obtained by training from the complete dataset) run independently by the cloud nodes was used as a comparison. The test set continued to use equal proportions of samples from each data subset, and the comparison experiment was repeated by adjusting the length of the generated tweets. The accuracy of R-AIBot was obtained from the weighted average of the accuracy of each node in proportion to the computation amount. As shown in Figure 16, R-AIBot did not perform as well as the local single-node-running LSTM model in terms of accuracy, but it could maintain a high level. The average time for tweet generation with R-AIBot gradually increased as the length of tweets increased, but there was no significant increase compared to the local LSTM model, so the delay due to inter-node communication was not significant.

$$Average\ Traffic = \frac{Traffic_{R-AIBot}}{number\ of\ nodes} \tag{17}$$

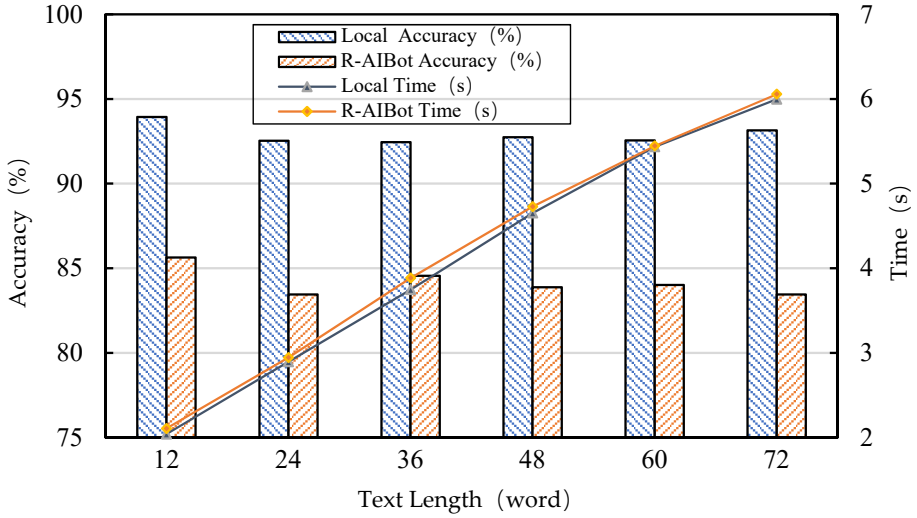

Figure 16. Variation in accuracy and execution time with text length.

The communication traffic in R-AIBot due to node co-computation was an important aspect of the performance evaluation. In this paper, we set different numbers of nodes and text generation lengths for multiple replication experiments, recorded the relevant data and calculated the average traffic of the nodes. The average traffic was obtained by dividing the total traffic generated in the network by the number of nodes, as given by Equation (17). To avoid interference from other factors, the same amount of computation was specified for each node in the experiment. As shown in Figure 17, the longer the length of the text generated, the more nodes were involved in the text generation and the higher the average inter-node traffic generated. Therefore, selecting an appropriate length of text generated and assigning fewer nodes to participate in performing intelligent computations were effective means to reduce communication traffic in R-AIBot.

5.3. AIBot Efficiency Evaluation

To evaluate AIBot as a botnet model, it was necessary to simulate and evaluate the execution efficiency of its launching attack activities. The attack execution time was the complete time from the initiation to the end of the attack, which, in AIBot, was equivalent to the time elapsed from the receipt of the computation task to the return of the result. The execution time of AIBot can be subdivided into command transmission time, program loading time, network reorganization time, model calculation time and result return time, which can be expressed as Equation (18). Program loading time is the time required for nodes to load program files, such as neural network models; network reorganization time

is the time required for the reorganization process of the network structure; and task computation time is the time required for each node to perform intelligent computation tasks. Table 1 lists four sets of scenarios for the experiments. C-AIBot classified only one set of samples at a time, R-AIBot generated text of length 12 each time, each set of experiments was repeated 20 times to take the average, and all were performed in the same experimental network (consisting of six edge nodes and one cloud node).

$$T_{AIBot} = t_C + t_P + t_N + t_M + t_R \tag{18}$$

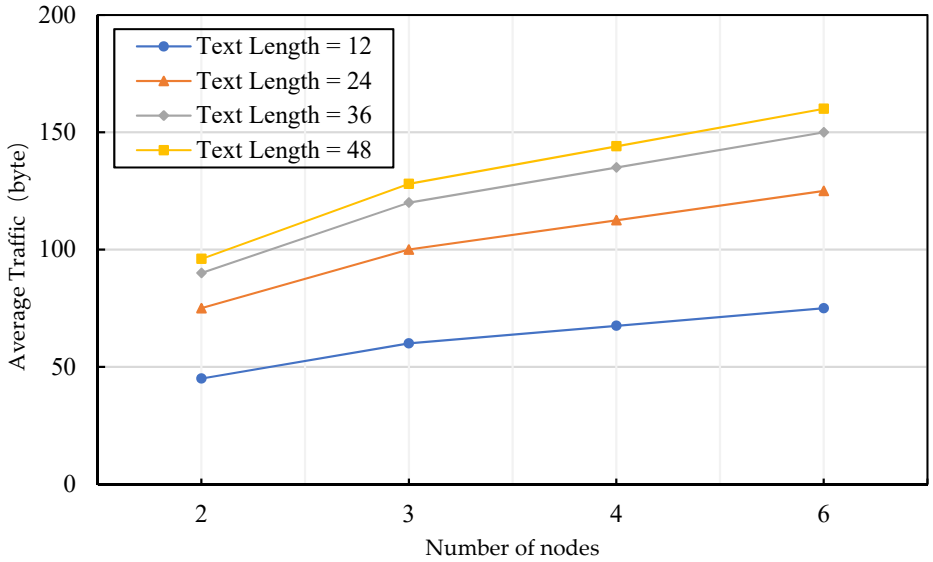

Figure 17. Variation in average traffic with the number of nodes for different text lengths.

Table 1. Experimental scenarios.

	Pre-Attack State	Type of Attack
1	Initial state	C-AIBot
2		R-AIBot
3	C-AIBot	C-AIBot
4		R-AIBot
5	R-AIBot	C-AIBot
6		R-AIBot

The results of the experiment are shown in Figure 18, where all subgroups were able to complete the assigned task within 15 s. The proportion of time spent on each of these items to the total time is shown in Table 2. Relatively speaking, program loading, network reorganization and model calculation accounted for a large proportion of the overall running time. Network reorganization took some time because different neural networks need the corresponding network organization to support distributed operations. The change of attack task led to a state switch in AIBot, thus requiring program update loading and network reorganization, and the execution time increased accordingly. R-AIBot took significantly longer than C-AIBot in both program loading and model computation because the LSTM model used in this paper was larger in size and computation than the CNN model. Overall, the running time of AIBot was as expected and the execution efficiency of C-AIBot was higher than that of R-AIBot.

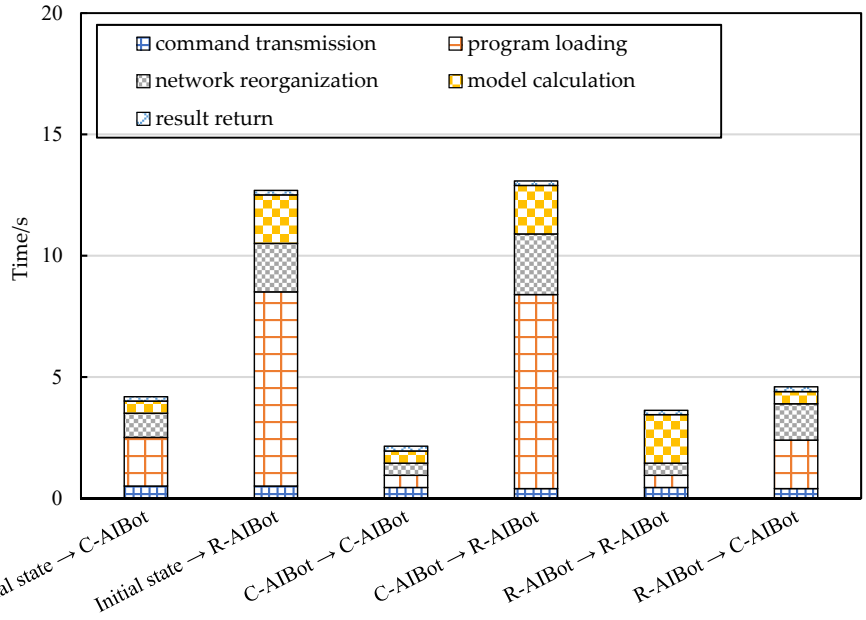

Figure 18. Execution time of AIBot.

Table 2. Percentage of time spent on each item.

	t_C (%)	t_P (%)	t_N (%)	t_M (%)	t_R (%)
1	12.17	47.73	23.87	11.93	4.30
2	4.02	63.04	15.76	15.77	1.42
3	20.93	22.27	24.25	23.24	9.30
4	3.06	61.16	19.11	15.29	0.14
5	12.40	13.76	13.78	55.10	4.96
6	8.70	43.48	32.61	10.87	4.35

6. Discussion

Based on the current network security situation, this paper proposes a botnet model that can perform intelligent computing in a distributed manner from the attacker's perspective and studies the construction mechanism and implementation methods. A new type of botnet similar to AIBot is likely to emerge in future cyber attacks and pose a great threat to cybersecurity. The purpose of this paper is to increase understanding of the new botnet by conducting relevant research ahead of the attacker, so that effective defense strategies can be better proposed. The following sections analyze the limitations of AIBot and discuss possible means of defense.

6.1. Limitations

It should be noted that the work in this paper belongs to academic research, and the proposed AIBot botnet, as a theoretical model, does not fully consider its applicability in the actual cyber adversarial environment and has a large gap with realistic botnet cases. The main purpose of this paper is to direct the attention of security researchers to this type of potential cybersecurity threat.

AIBot has the following inherent limitations: (i) AIBot can only run deployed intelligent computing models, and the training and optimization of the models are undertaken offline, as distributed online training requires massive data support, generates a large amount of communication traffic between nodes and continuously occupies node computing resources. (ii) AIBot supports only two neural network models, the CNN and RNN, while practical application scenarios have demands for all kinds of intelligent algorithms, such as using reinforcement learning for decision making, and different intelligent algorithms require different ways of network structure. (iii) The distributed intelligent

computing executed by AIBot has version requirements for hardware and software, and the experiments in this paper were conducted on nodes with complete environment dependencies installed, while an actual application would require custom development and code rewriting of the program. In summary, we believe that AIBot is still inadequate in terms of model optimization, function expansion and practical deployment, and is currently only at the proof-of-concept stage and not yet ready for large-scale application.

6.2. Defense Analysis

This paper studies potential new attack patterns for botnets and provides insight into their theoretical foundations and implementation mechanisms so that we can carry out relevant proactive defense work ahead of attackers. Although the AIBot model proposed in this paper enhances the adversarial properties in terms of program size, communication traffic and resource consumption, it is still not completely immune to all defenses. Potential defensive strategies are discussed in the following areas:

- **Based on bot program**: (i) Although AIBot simplifies and compresses the computational model, the bot program still contains a much larger number of parameters than ordinary programs, which can be exploited by defenders for static identification of malicious code. (ii) The computation process of CNNs and RNNs contains a large number of matrix operations, and it is difficult to use similar large-scale parallel computation in ordinary programs, which can be exploited by defenders for dynamic analysis of malicious codes;
- **Based on traffic analysis**: (i) A large amount of interactive communication between nodes occurs during the task execution in AIBot. The intermediate results of the intelligent computation are mainly passed on in the network in the form of probability vectors and text vectors, and the defender can characterize this abnormal communication traffic. (ii) AIBot performs intelligent computing in the form of a subnet of the botnet, and the nodes communicate with each other with certain laws. The defender can extract the network structure and, thus, infiltrate the botnet by mining the connection relationship between the nodes;
- **Based on terminal behavior**: Although AIBot uses optimized node computation, intelligent computation based on neural networks will inevitably generate transient high-resource occupancy, so continuous monitoring of hardware resource occupancy, detecting abnormalities and issuing alerts, is still a proven defense tool.

7. Conclusions

The security risks associated with artificial intelligence technologies are a hot topic of research [29,30]. In this paper, we study a new botnet attack mode resulting from combining the basic platform of cyber attacks with artificial intelligence technology, which, to some extent, reflects the cyberspace security risk caused by the misuse of artificial intelligence technology. In order to be prepared for future botnets, this paper predicts new botnet attack techniques from the attacker's perspective and proposes AIBot, a botnet model that can perform intelligent computing tasks in a distributed manner. AIBot deploys the neural network model in a distributed manner in the botnet and uses the idle computing resources of large-scale zombie nodes to perform intelligent computing tasks. Compared with the method of sending sensitive data obtained from end devices back to the cloud server and then processing them, AIBot appears to be more advanced and efficient and can pose a serious threat to cyberspace security. The next step is to explore the possibility of distributed online training of neural network models in botnets from the attacker's perspective and to propose practical defense methods against such botnets from the defender's perspective.

Author Contributions: Conceptualization, H.S. and H.Z.; methodology, H.Z.; software, H.Z. and J.Y.; validation, H.Z., J.Y. and Y.H.; data curation, H.Z., J.Y. and Y.H.; writing—original draft preparation, H.Z.; writing—review and editing, H.Z.; supervision, H.S.; funding acquisition, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China, grant number 2019QY1305.

Acknowledgments: We would like to thank the anonymous referees for their helpful comments and suggestions.

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

References

1. Vormayr, G.; Zseby, T.; Fabini, J. Botnet communication patterns. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2768–2796. [CrossRef]
2. Fang, B.; Cui, X.; Wang, W. Survey of botnets. *J. Comput. Res. Dev.* **2011**, *48*, 1315.
3. Seymour, J.; Tully, P. Weaponizing data science for social engineering: Automated E2E spear phishing on Twitter. *Black Hat USA* **2016**, *37*, 1–39.
4. Antonakakis, M.; April, T.; Bailey, M.; Bernhard, M.; Bursztein, E.; Cochran, J.; Durumeric, Z.; Halderman, J.A.; Invernizzi, L.; Kallitsis, M.; et al. Understanding the mirai botnet. In Proceedings of the 26th USENIX security symposium (USENIX Security 17), Vancouver, BC, Canada, 23 May 2017; pp. 1093–1110.
5. Dange, S.; Chatterjee, M. IoT Botnet: The largest threat to the IoT network. In *Data Communication and Networks*; Springer: Singapore, 2020; pp. 137–157.
6. Zhang, J.; Zhang, R.; Zhang, Y.; Yan, G. The rise of social botnets: Attacks and countermeasures. *IEEE Trans. Depend. Secur. Comput.* **2016**, *15*, 1068–1082. [CrossRef]
7. Ferrara, E. "Manipulation and abuse on social media" by Emilio Ferrara with Ching-man Au Yeung as coordinator. *ACM SIGWEB Newsl.* **2015**, *(Spring)*, 1–9. [CrossRef]
8. Casenove, M.; Miraglia, A. Botnet over Tor: The illusion of hiding. In Proceedings of the 2014 6th International Conference on Cyber Conflict (CyCon 2014) IEEE, Tallinn, Estonia, 3–6 June 2014; pp. 273–282.
9. Anagnostopoulos, M.; Kambourakis, G.; Drakatos, P.; Karavolos, M.; Kotsilitis, S.; Yau, D.K.Y. Botnet command and control architectures revisited: Tor hidden services and fluxing. In Proceedings of the International Conference on Web Information Systems Engineering, Puschino, Russia, 7–11 October 2017; Springer: Cham, Switzerland, 2017; pp. 517–527.
10. Fajana, O.; Owenson, G.; Cocea, M. Torbot stalker: Detecting tor botnets through intelligent circuit data analysis. In Proceedings of the 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 1–3 November 2018; pp. 1–8.
11. Li, K.; Fang, B.; Cui, X.; Liu, Q. Study of botnets trends. *J. Comput. Res. Dev.* **2016**, *53*, 2189.
12. Liu, W.; Wang, Z.; Liu, X.; Zeng, N.; Liu, Y.; Alsaad, F.E. A survey of deep neural network architectures and their applications. *Neurocomputing* **2017**, *234*, 11–26. [CrossRef]
13. Zhang, Z.; Yin, L.; Peng, Y.; Li, D. A quick survey on large scale distributed deep learning systems. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018; pp. 1052–1056.
14. Weaver, R. Visualizing and modeling the scanning behavior of the conficker botnet in the presence of user and network activity. *IEEE Trans. Inf. Secur.* **2015**, *10*, 1039–1051. [CrossRef]
15. Xie, Y.; Yu, F.; Achan, K.; Panigrahy, R.; Hulten, G.; Osipko, I. Spamming botnets: Signatures and characteristics. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 171–182. [CrossRef]
16. Plohmann, D.; Gerhards-Padilla, E. Case study of the miner botnet. In Proceedings of the 2012 4th International Conference on Cyber Conflict (CYCON 2012) IEEE, Tallinn, Estonia, 5–8 June 2012; pp. 1–16.
17. Shah, N.; Farik, M. Ransomware-Threats Vulnerabilities and Recommendations. *Int. J. Sci. Technol. Res.* **2017**, *6*, 307–309.
18. Berman, D.S.; Buczak, A.L.; Chavis, J.S.; Corbett, C.L. A survey of deep learning methods for cyber security. *Information* **2019**, *10*, 122. [CrossRef]
19. McDermott, C.D.; Majdani, F.; Petrovski, A.V. Botnet detection in the internet of things using deep learning approaches. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN) IEEE, Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8.
20. Yi, S.; Li, C.; Li, Q. A survey of fog computing: Concepts, applications and issues. In Proceedings of the 2015 Workshop on Mobile Big Data, Hangzhou, China, 21 June 2015; pp. 37–42.
21. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
22. Verbraeken, J.; Wolting, M.; Katzy, J.; Kloppenburg, J.; Verbelen, T.; Rellermeyer, J.S. A survey on distributed machine learning. *ACM Comput. Surv. (CSUR)* **2020**, *53*, 1–33. [CrossRef]
23. Teerapittayanon, S.; McDanel, B.; Kung, H.T. Branchynet: Fast inference via early exiting from deep neural networks. In Proceedings of the 2016 23rd International Conference on Pattern Recognition (ICPR) IEEE, Cancun, Mexico, 4–8 December 2016; pp. 2464–2469.
24. Teerapittayanon, S.; McDanel, B.; Kung, H.T. Distributed deep neural networks over the cloud, the edge and end devices. In Proceedings of the 2017 IEEE 37th international conference on distributed computing systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 328–339, IEEE.
25. Kim, Y.D.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; Shin, D. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv* **2015**, arXiv:1511.06530.

26. Yu, Y.; Si, X.; Hu, C.; Zhang, J. A review of recurrent neural networks: LSTM cells and network architectures. *Neural Comput.* **2019**, *31*, 1235–1270. [CrossRef] [PubMed]
27. Krizhevsky, A.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. In *Handbook of Systemic Autoimmune Diseases*; Elsevier Ltd.: Amsterdam, The Netherlands, 2009; Volume 1.
28. Trump_tweet_dataset. Available online: http://chirag2796.pythonanywhere.com/trump_tweet_dataset (accessed on 20 June 2022).
29. Radanliev, P.; De Roure, D.; Maple, C.; Ani, U. Super-forecasting the 'technological singularity' risks from artificial intelligence. *Evol. Syst.* **2022**, *13*, 747–757. [CrossRef]
30. Radanliev, P.; De Roure, D.; Maple, C.; Santos, O. Forecasts on Future Evolution of Artificial Intelligence and Intelligent Systems. *IEEE Access* **2022**, *10*, 45280–45288. [CrossRef]