# DGA-based Intrusion Detection System using Federated Learning Method on Edge Devices

Nguyen Ngoc Minh, Pham Trung Hieu, Vu Hai, Nguyen Huu Thanh
School of Electronic and Electrical Engineering
Hanoi University of Science and Technology, Vietnam

***Abstract*—Cybersecurity is one of the most important tasks to secure a network. In traditional approaches, Network Intrusion Detection Systems (NIDS) are usually located on the Cloud, which always handle large amounts of data or are integrated into firewalls that detect malicious network traffic by extracting specific network features. Both solutions have their own disadvantages. In this paper, we proposed a method for detecting network intrusion at edge devices while not compromising privacy. The proposed system focuses on detecting malicious domain names generated to evade Intrusion Detection Systems (IDSs). We implemented a machine learning algorithm on edge devices and applied the Federated Learning as an approach for distributed intrusion detection. Additionally, we considered the heterogeneity of Cloud-Edge systems and experimented with different non-IID distributions of data among heterogeneous clients. The findings of this study indicate that the proposed system is capable of effectively detecting harmful behaviors, even without sharing local data with the central server. The performance of the proposed system is comparable to that of centralized and traditional techniques.**

***Index Terms*—Federated Learning, DGA, NIDS, Machine Learning, Cloud Edge**

## I. Introduction

The Cloud and Edge computing is the prominent architecture of IoT devices and the central cloud usually has dominant computing ability. There are many applications integrated into the cloud with various types of data stored in a distributed manner, leading to the risk of various cyberattacks such as DDoS, malware, or even zero-day attack. Specifically, malware seeks to invade, damage, or disable computers, computer systems, networks, tablets, and mobile devices, often by taking partial control over a device's operations. Modern malware nowadays relies on some remote connection with their attacker's C2 (Command & Control) servers to operate. Previously, such connections were easily blocked by using rules and blacklisting methods. This is why Domain Generation Algorithm (DGA) was developed. DGA is a type of algorithm that takes a random seed, feed it through some algorithms to generate pseudo-random domains, and then concatenates it with an additional top-level domain (TLD). The attacker can ship the malware with the DGA algorithm to automatically generate domains and query it to the DNS server, eventually coinciding with the C2 server domain, thus the malicious connection is established. A traditional approach to block this type of threat is to use a signature–based detection system that contains a huge collection of signatures (Indication of Compromise) that a cybersecurity engineer could find via malware affected devices. Such a solution is reaching its limit because attackers always change the signatures of cyberattacks using complicated algorithms. In many cases, the cloud has to deal with numerous hardware-demanding tasks: computing, processing requests from a lot of edge devices, and pattern matching for the IDS. When the central cloud is overloaded, the performance of the cloud network might be reduced. Furthermore, mitigation applied after an attack has reached the central system is no longer an optimal strategy because the attackers have already achieved their goal. Thus, early prevention, close to the source of the attack, is required.

On the other hand, by making use of recent approaches based on machine learning such as deep learning, one can leverage the huge amount of data from multiple edge devices to learn complex hidden features that may distinguish between benign and malicious activities. However, traditional training methods require data to be gathered and learned in a centralized manner. Such requirements put a heavy burden on the network since the system has to consistently collect raw data from multiple clients. When the cloud edge system is scaled horizontally, the traffic load is increased accordingly and thus, the central server will undoubtedly be overloaded. Another drawback of the centralized approach is that clients' local data must be transferred to the server, therefore exposing themselves to the risk of leaking highly sensitive data, and infringing one's privacy. To address such issues, Federated Learning (FL) [13] is adopted to enable large-scale aggregation, allowing a large number of edge devices to share their 'knowledge' without exposing one's personal or proprietary data.

The contribution of our research are as follows:

1) We implemented a system that detects network intrusion, particularly DGAs, by classifying benign and malicious edge devices based on their domain queries. We employed a deep neural network that specifically targets character-based DGAs and allows edge devices to aggregate their experiences without sharing their local data using Federated Learning. Thus, we can build a globally high performance neural network for DGA detection that can serve a large number of edge devices.
2) We perform experiments to show that the proposed system's performance is on par with traditional methods of detecting DGAs, while still maintaining data privacy.
3) Based on the results of the experiments, we also discuss about the problem of data heterogeneity and its impact on the performance of an FL system.

## II. Related work

### A. Domain Generation Algorithms

DGAs nowadays are getting more and more complex in order to combat recent feature-based DGA detection mechanisms. The first malware used DGA to bypass domain blacklisting was Kraken in 2008. Kraken's randomly generated domains were generated using random characters, therefore it was easy to detect them using human intuition, i.e., ghcxn-cadnj.dyndns.org. Such DGAs were classified as character-based DGAs, where the domains are constructed using non-contextualized characters. Recent malware such as Gozi, Matsnu, or Suppobox [9], crafted their domains using a list of real words (dictionary-based), resulting in domains that are extremely challenging to recognize using human intuition, not mentioning automatically recognizing them using statistics or features. The latter was called dictionary-based DGAs.

There have been already a few proposed defensive mechanisms to combat the use of DGAs, further improving the effectiveness of anti-malware systems. An early approach was analyzing network logs to identify abnormal domain queries using statistical or hand-crafted features. Yadav et al. [21] proposed statistical measures such as Kullback-Leibler divergence, Jaccard index, and Levenshtein edit distance of uni-gram and bi-gram followed by a L1-regularized linear regression model to classify attack and benign (white-listed) domains. Bilge et al. [4] proposed the EXPOSURE, a framework that performs passive analysis of the DNS traffic, extracting 15 domain name features and then perform classification using the J48 Decision Tree classifier. Later in 2017, A. Ahluwalia et al. [2] extracted information theoretic features such as entropy, length, vowel rate, etc. from the domains before feeding them to two different supervised classifiers: Random Forest and Decision Tree. These early approaches require feature engineering and thus do suffer from model drifting, where newer malware build more complex domains.

Deep learning approaches have proven to be very effective against character based DGAs, but not very much for newer types of DGAs, where they use words from dictionaries and the crafted domains resemble English words. Such DGAs are called dictionary based DGAs. Curtin et al. [5] suggested a notion for measuring the complexity or difficulty of a DGA's family domain, called smash-word score, showing how much a domain looks like English words. At the same time, they add additional domain registration side information along with the domain to a Recurrent Neural Network and achieved reasonable results. Highnam et al. [10] created a novel hybrid neural network by using CNN and LSTM in parallel, then calculating the likelihood of such domain being malicious by bootstrap aggregating (or namely bagging).

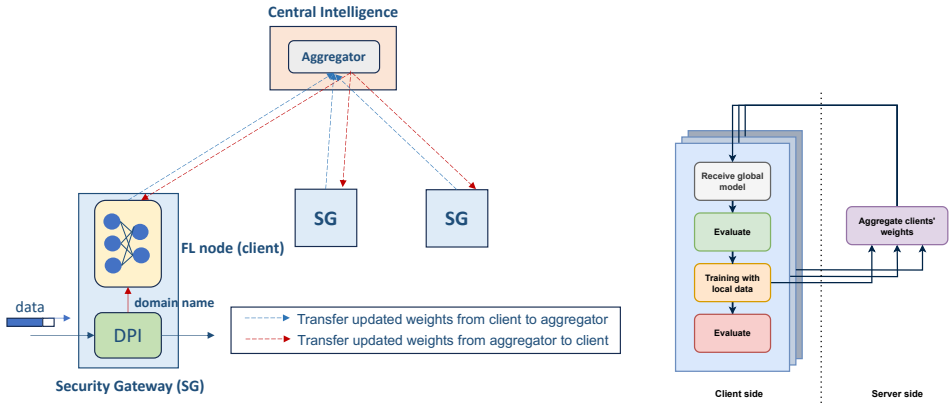
### B. Distributed Artificial Intelligence

Distributed Artificial intelligence (DAI) is a type of AI system that is designed to operate across multiple computers or devices, often in a distributed or decentralized manner. DAI has been an emerging field of research for more than 30 years, but not until recently it gains significant attention thanks to the increasingly large and hardware-demanding machine learning models, as well as bombarding amount of data. Such advances in technology are being bottlenecked by the requirement of hardware resources. By utilizing distributed systems, one can provide AI a larger degree of accessibility and scalability.

When considering distributing AI to multiple devices, there are two fundamentally different approaches: *horizontal* and *vertical* distribution. Distributing horizontally, sometimes called the Data Parallelism [12], is to partition the data into multiple parts and distribute them to multiple clients for processing. Ideally, each client receives a partition of the data, optimize their localized model using that data and then aggregate the model with other clients via a central server (sometimes called a Parameter Server). In Data Parallelism, each client has access to the same model via either cloning the global model or through some centralizing mechanism. This technique is straightforward to implement, however, based on the assumption that the data of multiple clients is independent and identically distributed (IID). The other approach to DAI is of partition AI vertically, which is called Model Parallelism [6]. In this method, each client should have access to the exact copy of the entire dataset, albeit operating on different parts of the model. This method performs best in the environment of multiple devices that share very high bandwidth connections (i.e. data center) due to the extensive amount of data to be transferred between the forward and backward propagation of the machine learning model.

In this paper, we discuss a method called Federated Learning (FL) [13], in some ways very similar to Data Parallelism. However, whereas Data-Parallelism focuses on distributing the data to clients for offloading the processing task, Federated Learning prioritizes working on the local data collected from each client, thus preserving the clients' data privacy. Since Federated Learning neither allows one client nor any central server to access another client's dataset, the assumption of IID properties shared among the clients can not always be satisfied [8]. There are multiple FL frameworks and tools such as Tensorflow Federated by Google [18], Flower [7], .etc. Previous work has successfully applied Federated Learning to solve real-world problems from many fields such as medical and healthcare [14], recommendation system [1], edge computing [15], .etc. However, there are still many problems those challenges the the application of Federated Learning in real-world scenario such as *security threats* [16], *heterogeneity* [17] and *communication cost* [22].

As previously addressed, in order to effectively counteract attacks and malicious activities in cyberspace, machine learning-based methods usually require that user's data should be shared, which could cause privacy issues. To the best of our knowledge, our proposed system is the first to apply the FL architecture to solve the problems of DGA detection. Inherently, domain query data is a rather sensitive and personalized data. The deployment of FL in this context proposes the sharing of knowledge between multiple devices, thus improve the effectiveness of security systems by enabling them to process




(a) FedAverage Topology in Security (b) System Dataflow

Fig. 1: High-level FL architecture

and analyze a large amount of data in real-time, meanwhile, still maintaining data privacy. At the same time, experimental results show that the performance of our proposed model do not suffer compared to conventional centralized machine learning approaches in terms of training a DGA detection model.

## III. Methodology and Implementation

Federated learning is a machine learning approach that enables multiple parties to train a shared model without sharing their data with each other. Our work adopted the *Federated Average* (FedAvg) architecture from H. Brendan McMahan et al. [13]. In FedAvg, each party (also known as a "node" or a "client") trains a model on its own data and then shares the model parameters with a central server or aggregator. The aggregator then combines the model parameters from each node and uses them to update the shared model. Figure 1a shows the overall topology of a FedAvg system deployed in security context. As shown in the figure, conventional security gateways (SG), such as firewalls are usually connected to a Central Intelligence that collects data for analysis. In order to implement a distributed DGA detection mechanism, we suggest that a FL node is integrated in each SG. The FL node that runs an ML model collects the domain names from the deep packet inspection (DPI) component. The ML model parameters in the FL node are then shared with the aggregator located in the Central Intelligence. Figure 1b illustrates the high-level procedure's dataflow of FedAvg. Generally, an FL procedure optimizes a global model by iterating the following process: 1) The aggregator distributes the global model's weight to the nodes that participate in the FL process, 2) The node then optimizes its model using some optimization algorithm and its locally collected data, 3) The aggregator then collect nodes' models, use some aggregation scheme to combine the nodes' model before updating the global model and begin the new training round.

To detect client infected with malicious software, we implemented a Long-Short Term Memory (LSTM) model from the previous work of Highnam et. al. [11] as a binary classifier to classify malicious versus benign domain names. LSTM is a improved version of Recurrent Neural Network (RNN). LSTM has advantages in handling data containing temporal information such as combination of characters and maintain long-term dependencies for longer domain names. Such model have proven to be very effective in classifying character-based DGAs [11] and also lightweight, which is suitable for the FL system where the neural network model must be replicated to train and do inference at each edge device. We did research on data published on Umbrella Popularity List [19] for top 1 million most popular websites, and found some common features about benign domains. They usually have meaningful words in their domain name rather than random characters. These type of DGAs require significant improvement in machine learning model to differentiate DGA and benign domain, which in turn demand higher computing power on edge devices. In our research, we aim solely on character-based DGA rather than the counterpart word-based DGA. We employed the algorithms collected in *baderj* [3] available from the Github repository. This *baderj* [3] repository provides a list of synthetic algorithms for DGAs, which were reverse engineered by security researchers. The process of constructing such algorithms is out of scope and will not be discussed in this work. We then use traditional training method and visualize the cluster of domains' embeddings as figure 2 to prove that the LSTM model can properly distinguish the DGAs and benign domains given that the model has been trained sufficiently. The training objective for LSTM model is the same: binary classification. The embeddings we used are collected from the output layers of the LSTM model, which is 128-dimensional. We then used a T-Distributed Stochastic Neighbor Embedding (t-SNE) [20] implementation from scikit-learn in Python to reduce the dimensionality of your 128-dimensional embeddings to a 2-dimensional space. T-SNE is a stochastic algorithm, and the distances between points on the plot do not necessarily reflect the original distances in the high-dimensional space. However, it realize the randomness

**Algorithm 1:** Non-IID sampling

Algorithm parameters: IID rate $r_{IID}$, number of sampled data $N_s$, sets of $K$ different data types $\mathcal{C}_1, \mathcal{C}_2, ...\mathcal{C}_K \subset \mathcal{C}$, skewed data type $k$.

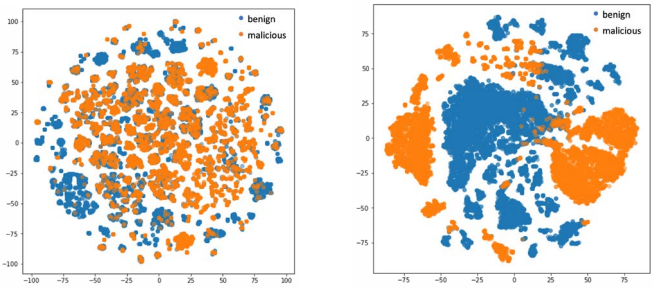
**Function** `sample-non-iid`$(r_{IID},\ N_s,\ k)$**:**

- $\mathcal{S} \leftarrow \emptyset$
- **for** $j = 1, 2, ...K$ **do**
  - **if** $j \equiv k$ **then**
    - $\mathcal{S} \leftarrow \mathcal{S} \cup$ (randomly sample $(1 - r_{IID}) * \|\mathcal{C}_j\|$ data points from $\mathcal{C}_j$)
  - **else**
    - $\mathcal{S} \leftarrow \mathcal{S} \cup$ (randomly sample $r_{IID} * \|\mathcal{C}_j\|$ data points from $\mathcal{C}_j$)
  - **end**
- **end**
- **return** $\mathcal{S}$

pattern of the embeddings before and clusteried pattern after training.

## IV. Experiments

Our experiments are based on synthetic simulation, in which different data are fed to a number of clients deploying federated learning. In the experiments we did not take into consideration the difference of processing and network capability among clients and only focus on the features of the dataset. To evaluate the performance of the proposed approach, we choose binary classification task using a customized dataset and we choose the Receiver Operating Characteristic (ROC) curve and Area Under Curve (AUC) score to evaluate the performance of the aggregated model. AUC measures the ability of a binary classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the model's performance in terms of distinguishing between the positive and negative classes. We tested the system based on three main test strategies: 1) Using the same number of clients participate in the FL process, but different amount of training data at each client. We use this test case to investigate *the impact of the amount of training data on the performance of the aggregated model*; 2) Using the same amount of training data at each client but different in the number of




(a) Before training (b) After training

Fig. 2: Clusters of domains visualized using TSNE

TABLE I: FL system simulation parameters

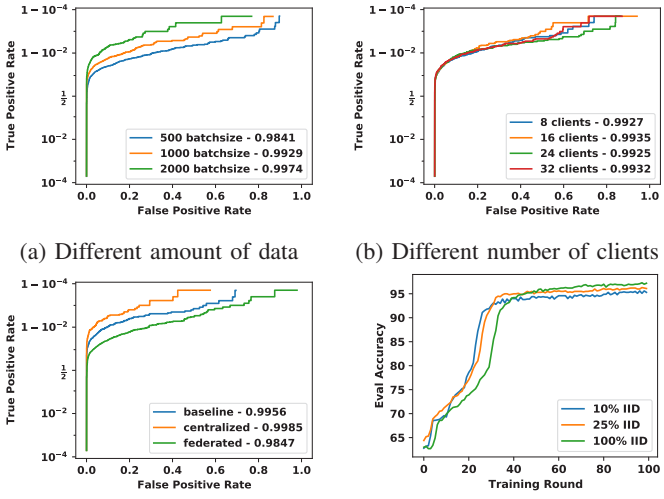
| Parameter name | Values |
|---|---|
| Number of clients | 8, 16, 24, 32 |
| Number of DGA distributed to each clients | 500, 1000, 2000 |
| Number of communication rounds | 100 |
| Number of local epochs | 1 |
| Local batch size | 100 |
| IID-rate | 0.1, 0.25, 0.5, 1 |

clients participating in the FL process. This test reflects *how horizontally scaling the FL system impacts the performance of the global model*; 3) Using same amount of training data overall but different implementation of training: centralized and federated learning. This is to see *how distributed training performs compared with traditional centralized methods*; 4) Using the the same number of clients, amounts of data, and other hyperparameters except for the distribution of the data at each clients. Our goal in this experiment is to *simulate the problem of data heterogeneity and to which degree data heterogeneity affects the performance of the aggregated model*. In our first 3 test strategies, we explicitly use Identically and Independently Distributed (IID) data at each client, and only in test strategy 4 we introduce the non-IID distribution (under our synthetic non-IID sampling methid). Centralized approach in our experiment means that the server should collect the data from all the clients and then train the model locally sever side using the collected data. In our test, we consider the most trivial implementation of Centralized Learning (CL), where one client simulate data generation and then send the data to the sever each round. We also adopt the conventional training model, where all rounds' data is gathered and training in one round and we consider this as a *baseline* for comparison. Other evaluation will be compared with this baseline to illustrate the effects of FL process in multiple cases. Our complete simulation parameters and their legend explanations used in Fig. 3 are shown in Table I.

TABLE II: Number of samples w.r.t. DGA families

| DGA family | Number of samples |
|---|---|
| corebot | 124961 |
| fobber | 125283 |
| kraken | 71673 |
| locky | 125316 |
| murofet | 125500 |
| newgoz | 124683 |
| qadars | 124833 |
| zloader | 58322 |

Our dataset consists of domain names and their corresponding labels: *malicious domains* (DGA domains) and *benign domains* (legitimate domains). The DGA domains were chosen from collections of related DGAs, known as DGA families, generated by reverse engineering a set of DGA malware [3]. Several families of DGAs derived from this source were empirically identified as character-only DGAs based on the structure of the domain names generated by the malware




(a) Different amount of data

(b) Different number of clients

(c) Different methods

(d) Different IID-rates

Fig. 3: Performance comparison between different configurations and approaches

samples and the algorithm used to generate those DGAs. The *corebot, fobber, kraken, locky, murofet, newgoz, qadars, zloader* families were chosen. After eliminating duplicate domain names, Table II provides the number of DGA samples collected with respect to each DGA family. The training set's legitimate domains originate from the Cisco Umbrella top 1 million domains [19]. After collecting all the domains and their labels, we randomly divide the data into two sets: the training set and the validation set with the proportion of each being $0.9$ and $0.1$ respectively. Primarily, the training set is used to sample the training data and distribute them to the clients in our FL process following the aforementioned 3 test strategies. The validation set is used by the central server to validate the efficacy of the aggregated model at each and every communication round. We employed Algorithm 1 to sample the data at client side. For instance, $r_{IID} = 0.3$ means that when providing data to each client in our simulation, we sample $30\%$ of the data randomly over all labels, and the other $70\%$ are sampled in a way that all of them have identical labels. In this algorithm, we create a mapping of individual client versus a single DGA family. Multiple clients can share the same DGA family according to their mappings, and that DGA family is the majority in the client's local dataset. This allows us to address not only the extreme IID and non-IID cases but also the cases, in which the data from each client are partially IID.

As shown in Figure 3a, we have experiment with different amount of data that synthetically distributed to each clients ($500$, $1000$ and $2000$), as in Table I. In the labels, 500b indicates that we have distributed to each client in the FL process 500 domain names, all of which are distributed in IID manner. It is shown from the graph that by increase the amount of data in the client side, the overall performance of the aggregated model is better accordingly. Whereas, in Figure 3b, where we try to differentiate the number of clients ($8$, $16$, $24$ and $32$) and fix other parameters in Table I. Despite using more data, it yields little to none improvement on the quality of the aggregated model. This can be explained based on the process of batched model optimization. During such optimization process, the data are divided into multiple batches (the size of these batches at each client is called the local batch size) and then the optimizer performs model update iteratively for each batch of data. In our experiment, we deployed the same local batch size, in which results a higher number of batches for a higher amount of data. The higher number of batches means more optimization steps in one round, thus create a larger gradient stride when we aggregate all clients' model. As for scaling up the number of clients, the results is nearly the same because the same amount of data means gradient step from each client remain unchanged.

Our second experiment is to compares FL with CL and baseline approaches. The implement of CL and FL used the same data partitions from for each round. The difference is, in each round, the client in the CL scenario does not perform any training, but sends all data to the central server. The central server combines all the collected datasets into a single one for training. For the baseline, we disregard any notion of clients and de-centrality by collecting all the data from all clients and train the model in one single training loop. In this experiment, we used $8$ clients, each client had $500$ data points and trained for $100$ rounds. As depicted in Figure 3c, CL method is the one yields the best performance in term of AUC score. This can be explained by the model optimization process similar to the case in Figure 3b. In Figure 3b, we use different of number of clients to identify the impact of such variable to the overall performance of the model. Given that each client has the same amount of data, our hypothesis is that the performance of the FL algorithm does not depend on the number of clients. Our hypothesis is that given the similar amount of data at each client, the performance of FL method is invariant to the number of clients. The explanation is that each client contribute to the central model a gradient vector, scaling up the number of client only increase the number of vectors in the aggregation process yet the vector distribution remains unchanged as the data are IID. By collecting all clients' data, the central server have a significantly larger number of batches in each rounds, thus makes the aggregated model outperforms other two methods. As also observed in Fig. 3c, the baseline model performs better than FL as FL suffered from a phenomenon called gradient divergence [23]. This caused each locally optimized model from each client to diverge from each other, inherently conflict with each other and reduce the global aggregated model's performance. Overall, in this experiment, we can conclude that FL process indeed negatively impacts the final performance of the ML model. However, since the different between the centralized and federated version is only $0.0146$ in terms of AUC score, we consider this a small trade-off for the overwhelming advantage of data privacy.

The phenomenon of gradient divergence is further illustrated in our final experiment (Fig. 3d), where we introduced the

non-IID distribution of data to the proposed system. In this experiment, we generate non-IID datasets by randomly sample the data at each client with weight. The degree of IID for a client's dataset is reflected using a parameter called IID-rate, or IID %. Our strategy to sample the data is shown in Algorithm 1. In this experiment, we used 8 clients, training for 100 rounds and each client has a batch size of 1000. Our results shows that the global aggregated model's performance increase accordingly to the IID-rate. The best performer is the aggregated model from 100%IID sampled dataset while the worst is the 10% IID one. 100% IID means that the local dataset is completely distributed in an IID manner, which results in the gradient divergence being smaller than the case of non-IID distribution. Whereas, in the 10% case, each client only have 10% of their data randomly distributed, the rest 90% is skewed to one specific label. Because of the skewed-ness in the data, we can reason that different clients' gradients is not consistent. Therefore, the aggregated model is very unstable and does not perform well compare to higher IID percentage.

## V. Conclusions

This paper proposes an approach for DGA detection based on Federated Learning. The approach can be deployed in a distributed manner in edge devices or security gateways at users' sites. Previous DGA detection mechanisms require that the domain name records be collected and sent to a central entity, thus relying on a single system to perform inference. By using FL, we allow the central intelligence to learn from all the edge devices while not sharing any sensitive data such as domain access record to the central entity. Since FL keeps raw data on the device and only sends model updates to the central server, FL itself is a layer of protection, reduces the risk of personal data leakage, and ensures the security of operations. On the other hand, FL allows the distributed system to use locally sensitive data to train and contribute to an aggregated model that distills the knowledge of a swarm of edge devices. The results of this study demonstrated that, despite not directly sharing local data with the central server, the proposed system can maintain reasonable performance of detection malicious activities, comparable with centralized and traditional techniques. Additionally, where as the number of clients does not affect the performance of the aggregated model, the amount of data distributed to each client and the IID-ness of the data matters. However, our test on data heterogeneous setting also showed that the proposed FL system are susceptible to the heterogeneity problem. Another problem need to be addressed is system heterogeneity, where multiple device can have different computational capabilities and each have a different IID-rate. It is our future work to implement a more robust FL schemes such as hierarchical client selection mechanism to address the problem of heterogeneous device and data.

## Acknowledgments

This research is funded by Ministry of Education and Training under project number B2023-BKA-10.

## References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] Aashna Ahluwalia, Issa Traore, Karim Ganame, and Nainesh Agarwal. Detecting broad length algorithmically generated domains. In Issa Traore, Isaac Woungang, and Ahmed Awad, editors, *Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*, pages 19–34, Cham, 2017. Springer International Publishing.

[3] Johannes Bader. Domain generation algorithm. https://github.com/baderj/domain_generation_algorithms.

[4] Leyla Bilge, Sevil Sen, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. Exposure: A passive dns analysis service to detect and report malicious domains. *ACM Trans. Inf. Syst. Secur.*, 16(4), apr 2014.

[5] Ryan R. Curtin, Andrew B. Gardner, Slawomir Grzonkowski, Alexey Kleymenov, and Alejandro Mosquera. Detecting DGA domains with recurrent neural networks and side information. *CoRR*, abs/1810.02023, 2018.

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[7] Flower: A friendly federated learning framework. https://flower.dev/.

[8] Dashan Gao, Xin Yao, and Qiang Yang. A survey on heterogeneous federated learning, 2022.

[9] J. Geffner. *End-To-End Analysis of a Domain Generating Algorithm Malware Family*, 2013.

[10] Kate Highnam, Domenic Puzio, Song Luo, and Nicholas R. Jennings. Real-time detection of dictionary DGA network traffic using deep learning. *CoRR*, abs/2003.12805, 2020.

[11] Kate Highnam, Domenic Puzio, Song Luo, and Nicholas R. Jennings. Real-time detection of dictionary dga network traffic using deep learning, 2020.

[12] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[13] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

[14] Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Medmcqa : A large-scale multi-subject multi-choice dataset for medical domain question answering, 2022.

[15] Sharnil Pandya, Gautam Srivastava, Rutvij Jhaveri, M. Rajasekhara Babu, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Spyridon Mastorakis, Md. Jalil Piran, and Thippa Reddy Gadekallu. Federated learning for smart cities: A comprehensive survey. *Sustainable Energy Technologies and Assessments*, 55:102987, 2023.

[16] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.

[17] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–17, 2022.

[18] Tensorflow federated: Machine learning on decentralized data. https://www.tensorflow.org/federated.

[19] Umbrella popularity list. http://s3-us-west-1.amazonaws.com/umbrella-static/index.html.

[20] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[21] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A.L. Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. IMC '10, page 48–61, New York, NY, USA, 2010. Association for Computing Machinery.

[22] Guang Yang, Ke Mu, Chunhe Song, Zhijia Yang, and Tierui Gong. Ringfed: Reducing communication costs in federated learning on non-iid data, 2021.

[23] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. 2018.