# Malware Network Traffic Classification on the Edge

1st Eric Chen
Computer and Information Science and Engineering
University of Florida
Gainesville, USA
echen2@ufl.edu

2nd Alexander Perez-Pons
Electrical and Computer Engineering
Florida International University
Miami, USA
aperezpo@fiu.edu

Abstract—Network traffic classification is a part of many cybersecurity applications, such as intrusion detection systems and anomaly detection. Currently, many cybersecurity tasks employ a cloud computing architecture, including network traffic classification. However, this architecture may not be able to meet the latency demands in the future with the ever-increasing number of network data. Therefore, we propose to perform network traffic classification utilizing edge computing and applying tiny machine learning to classify closer to the data source. Our approach uses TensorFlow Lite to convert a traditional convolutional neural network into a tiny machine learning model that runs on an edge device. In order to access the impact of edge classification, we ran simulations on the edge device and compared these results to a cloud-computing counterpart. We determined that the edge device was faster and had reduced latency compared to a cloud computing alternative, with a negligible reduction in accuracy throughout all experiments.

Index Terms—Traffic Classification, Convolutional Neural Network, Edge Computing

## I. Introduction

Network traffic classification is the process of determining the type of traffic that occurs on a specific network. It is a crucial and fundamental component of modern-day cybersecurity, providing functionality to services such as quality of service (QOS) control, anomaly detection, and intrusion detection systems [1]. Currently, network traffic classification is usually done through machine learning techniques, specifically deep learning techniques. Much research has shown that utilizing deep learning techniques, such as convolutional neural networks (CNNs) and Long Short-Term Memory (LTSM), can efficiently and accurately perform network traffic classification [2].

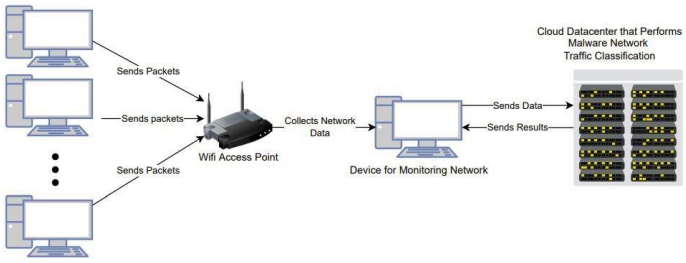


Fig. 1. Cloud-based architecture of malware network traffic classification

There are many approaches to conducting network traffic classification using machine learning models, but the most common one is to use a cloud computing architecture. In this architecture, network data is locally captured, then sent to a server on the cloud that actually performs the classification. Figure 1 displays an example of a cloud computing architecture for network traffic classification. While this architecture is sufficient for current needs, it may not be sufficient to meet future demands, given the forecasted growth of network devices. Statista reports that the number of Internet of Things (IoT) devices will drastically increase as time goes on, more than doubling by 2030 [3]. As a result, network traffic will exponentially increase, causing bottleneck issues for a cloud computing architecture. The increasing amount of data needing to be sent to the cloud and receiving results will cause lag and increase the delay in obtaining results [4]. The increase in delay will impair the functionality and efficiency of many cybersecurity systems, especially for new advancements like autonomous vehicles that require results quickly.

One solution to combat delays is to convert from a cloud computing structure to an edge computing structure. Edge computing processes data on edge devices, devices close to the data source that controls data flow on the boundary, without interacting with the cloud. It allows edge computing many advantages over cloud computing with decreased response time, less bandwidth pressure, and real-time processing by reducing delays [5]. The field of edge computing that the paper focuses on is tiny machine learning (TinyML). TinyML focuses on allowing microcontrollers with low processing power and memory to use machine learning techniques and algorithms to perform tasks [6]. It has the same benefits as cloud computing with reduced latency. Furthermore, there is support to easily convert traditional machine learning models used for applications like malware network traffic classification to their tiny counterparts that edge devices can use. The main goal of this paper is to establish the feasibility and benefits of converting a traditional malware network classification model to a tinyML model and having it run on an edge device.

This paper uses TensorFlow Lite, a popular TinyML framework, to achieve the benefits of edge computing. The original model created was a CNN that was trained using the USTC-TFC2016 dataset by Keras [7]. It was then post-training integer quantized and converted to a TensorFlow Lite model using built-in TensorFlow functions and resources. The average accuracy of both models was tested, with both models seeing 100% average accuracy. Furthermore, compared to a cloud

computing architecture with the initial model, the TensorFlow Lite model on an edge device was faster at every input size.

## II. Related Works

Three main methods have been researched and implemented to perform network traffic classification: port-based methods, payload-based methods, and machine learning methods.

The first approach to doing network traffic classification uses port-based methods. They used predefined hard-coded rules, classifying based on mapping and checking ports used by well-known applications [8], [9]. However, this approach has become functionally obsolete in recent years due to the increase in applications using dynamic port allocation and sometimes even intentionally disguising their port numbers to avoid detection [8]–[12].

To overcome the deficiencies of port-based methods, researchers began investigating and implementing payload-based methods. Payload-based methods perform Deep Packet Inspection (DPI), a technique that examines packets to look for distinctive application characteristics for classification [9], [10]. While this advancement in traffic classification has shown promising results, it still has inherent flaws. Performing DPI requires expensive hardware and has high computational costs [8], [9]. Furthermore, DPI does not work on encrypted traffic, something becoming more prevalent in recent history [8]–[12].

These flaws in the previously mentioned methods led to modern research about utilizing machine learning techniques to perform network traffic classification. Machine learning techniques use flow statistical features that are port and payload-independent, allowing them to be more flexible while avoiding some problems that arise with port-based and payload-based approaches [11]. Much research has been done on this topic, especially concerning deep learning. In [7], they first proposed using a CNN to perform network traffic classification using raw network data. Later, Lim et al. [13] further expanded on this idea by proposing the use of ResNet, a CNN involving a shortcut connection, that saw promising results. Further research saw positive results for implementing recurrent neural networks (RNNs), sparse autoencoders (AEs), and generative adversarial networks (GANs) [14].

While research is being done on finding effective models for network traffic classification, fewer resources have been allocated to finding effective ways to implement them. The other main research on effectively implementing network traffic classification is from Kim et al. [15]. They proposed using a hybrid edge computing and cloud computing architecture to perform network traffic classification [15]. This paper takes inspiration from deep learning works like [7] to construct and show the effectiveness of an edge-computing architecture in practice that is similar to the serving phase mentioned in [15].

## III. Methodology

### A. Dataset and Preprocessing

In theory, a device that could collect network traffic data would collect raw data in bytes. To emulate real-time raw network traffic, we used the session data with every layer in the USTC-TFC2016 dataset. It saw the highest accuracy in [7] paper and satisfied the requirement of being raw data. Lastly, the successful results found by [7] and the similarities between our models led to the choice of using their dataset.

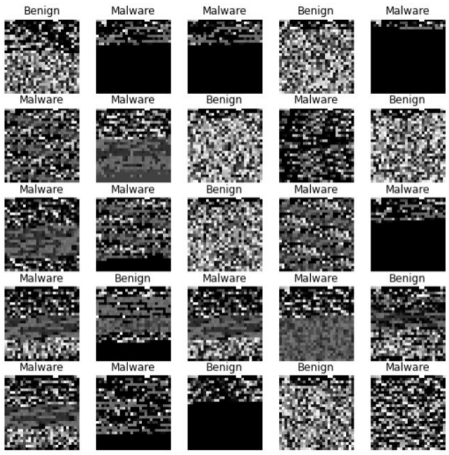


Fig. 2. Example images of network traffic

For preprocessing to be realistic for an edge device, it must take a low amount of memory and processing power. Fortunately, preprocessing raw network traffic data is very simple. The only preprocessing for an individual data point in this dataset is to shorten it to 784 bytes if it is over 784 bytes or append zeros to the end if it is less than 784 bytes. This is simple enough for an edge device to replicate with real-time data. However, when training and testing the execution times of the models, we used the already preprocessed data, avoiding the step of conducting preprocessing. Figure 2 provides images of the data after preprocessing as a 28*28 image.

### B. Model Architecture

As mentioned previously, the model architecture is very similar to the one proposed by [7], a CNN architecture like LeNet-5, with some more channels in each convolutional layer. However, there are fewer channels than the model proposed by [7] to lower the model size so that it can be used by an edge device more efficiently, given the edge device's inherent lack of memory and processing power.

Initially, we obtain traffic images of size 28*28*1 associated with each packet. The image pixels were not normalized to increase the accuracy of the tinyML model since it would be post-training integer quantized. Furthermore, the edge device used would only accept integer inputs. Therefore, by not normalizing the model, the model was trained to deal with integer inputs in the first place.

The model architecture has eight layers with 206,017 trainable parameters. The first convolutional layer has 16 filters with a 5*5 kernel size and a relu activation function. Then, there is a 2*2 max pool layer that performs a max pooling operation. Following that, there is another convolutional layer

with 32 filters and a 3*3 kernel size with a relu activation function. After that, there is another 2*2 max pool layer. Then the results are flattened to shift towards fully connected layers. The first fully connected layer has 128 outputs with a relu activation function. Then, there is a dropout layer with a rate of 0.5. The final layer is a fully connected layer with an output size of one with a sigmoid activation function. This is the model structure for the original Keras model used later to test the execution times between a cloud computing architecture and an edge computing architecture. Figure 3 shows the pipeline of the original model architecture.

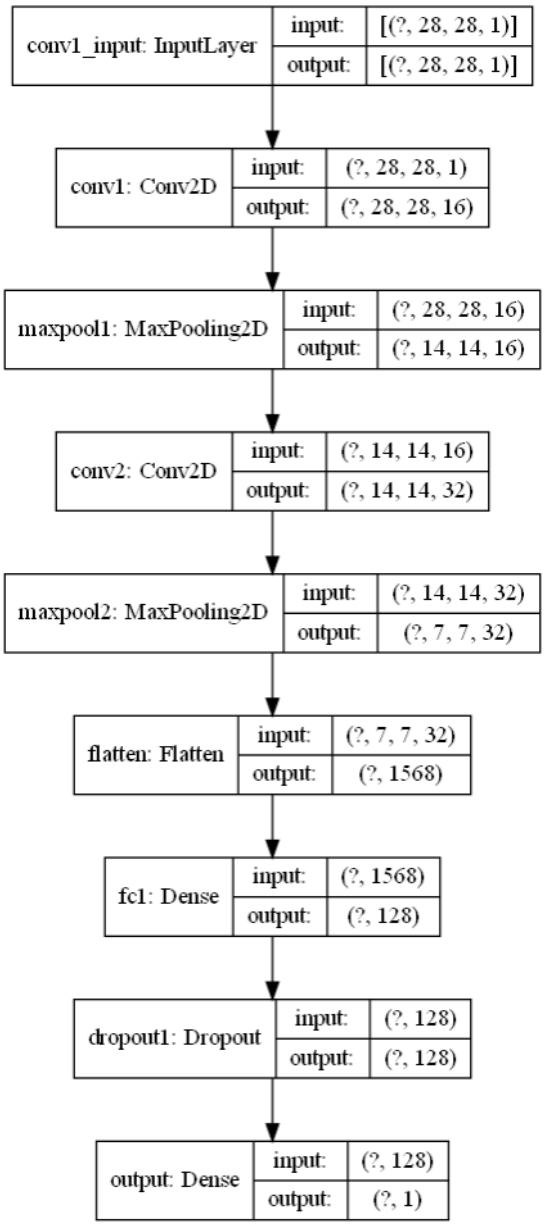


Fig. 3. Original model architecture

The original Keras model was then directly converted to a TensorFlow Lite model, which was optimized using the default parameters and post-training integer quantized as specified according to the TensorFlow Lite indications.

C. Hardware

Fig. 4. Picture of the WIO Terminal

To render an edge-computing architecture, we employed a WIO Terminal to run our tinyML model. This device has built-in support for tiny machine learning, has WiFi capabilities with an RTL8720DN, and embodies the low processing power and memory constraints typically associated with edge devices. It is powered by an ATSAMD51P19 microchip with an ARM Cortex-M4F core with a 4 MB external flash, 192 KB RAM, and 512 KB program memory size [16]. Figure 4 displays a picture of the WIO Terminal.

D. System Model

Throughout an area, there can be many WiFi access points. To implement malware network traffic classification, you could have an edge device that sniffs the packets of each access point. Then, the edge device could preprocess the packets, classify them using a model, and send the data to a monitoring station upon the detection of suspicious communications. Figure 5 illustrates an example of one access point.

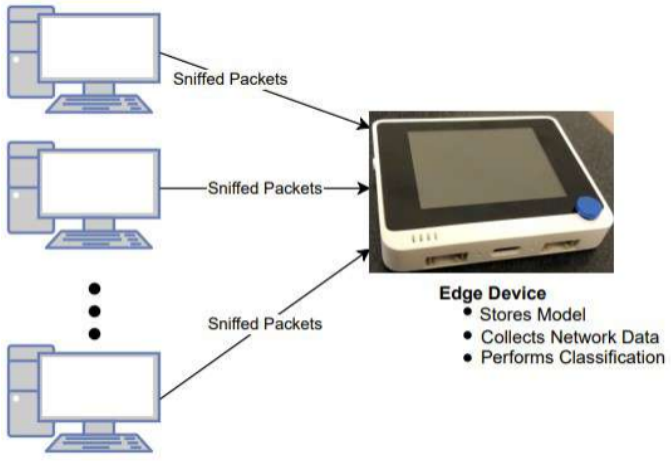


Fig. 5. Edge computing architecture for malware network traffic classification

## IV. Evaluation

### A. Experiment Setup

The environment used to create the integer-based TensorFlow Lite model and float-based Keras model used Python 3.8, TensorFlow 2.3, and Keras 2.4.3. The original Keras model was uploaded to the Google Cloud Platform and called through Cloud Run. It was hosted on the default execution environment with 16 GiB of memory and eight virtual CPUs.Then, the xxd command was used to convert the TensorFlow Lite model into a C array that an Arduino can use. The xxd command creates a hex dump of the TensorFlow Lite file, allowing an Arduino to read and use the model. After that, we uploaded the model to the WIO Terminal facilitated through the Arduino IDE and TensorFlow Lite Arduino library to have the WIO Terminal perform classification.

### B. Performance Metrics

Three evaluation metrics were used, consisting of the model size, model accuracy, and execution time. The model sizes were the file sizes provided in the development environment. Accuracy was calculated by taking the total number of test examples predicted correctly over the total number of test examples. The execution time measured the time to feed inputs into the model, use the model to get an output and print the label to a device.

### C. Experiment Results and Analysis

As seen in figure 6, the tinyML model is smaller than the original Keras model, only 25% of the size of the original model. Furthermore, figure 6 shows that the array representation of the tinyML model is smaller than the array representation of the original Keras model, roughly 25% the size of the array representation of the original Keras model. This shows that different representations of a tiny model will require less space to store than the same representations of their original model.

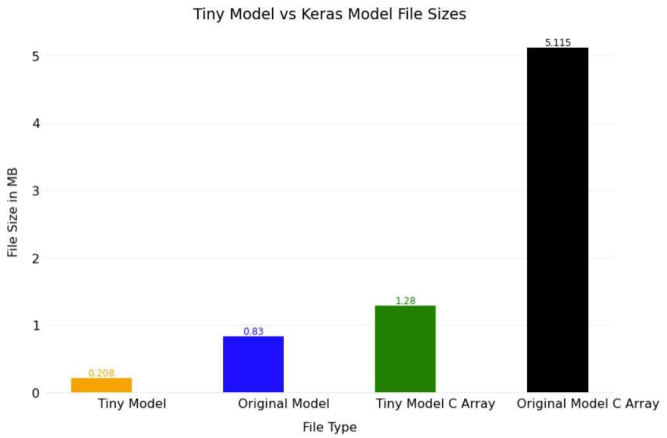


Fig. 6. File sizes of models in different forms

Both models saw an accuracy of 100%. Although the original model is slightly different, it is similar in nature to the original model architecture in [7], meaning it makes sense that the accuracy of the original model is basically the same as the accuracy reported in [7]. Even with the integer quantization and reduced complexity, the tiny model does not suffer from a severe drop in accuracy. This shows that the tinyML model approximates the original model effectively. Since the model accuracies are essentially the same, an edge device using the tinyML model would classify network traffic similarly to that model on the cloud.

Finally, as seen in figure 7, the edge-computing architecture saw a lower execution time than a cloud-computing architecture at every load. In the beginning, the execution times on the edge device are around 60% of the execution times for the cloud architecture. However, as the load size increased, the execution times of the edge device became even better compared to the cloud model, taking less than 50% of the time. This shows that doing network traffic classification on an edge device is not only possible but is faster than a cloud-based model as well.

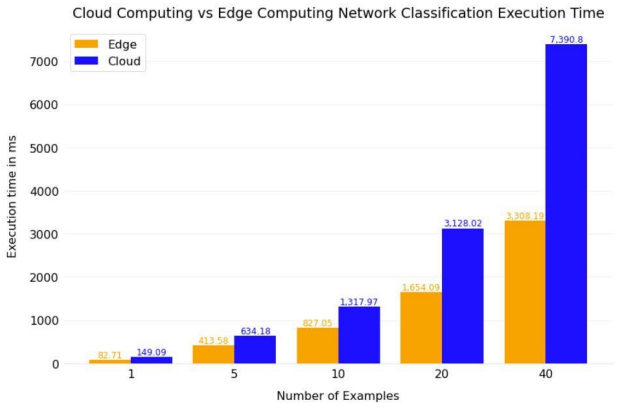


Fig. 7. Malware network classification execution times of an edge-based architecture versus a cloud-based architecture

## V. Conclusion

### A. Principal Findings

In order to prepare for the exponentially increasing amount of network data being transferred, we proposed using an edge-computing architecture to perform network traffic classification. We created a CNN model through Keras and converted it to a TensorFlow Lite model for an edge device to use. Through our experiments, we discovered that the accuracy of the TensorFlow Lite model traffic classification is sufficiently accurate to be used, not showing a relevant decrease in accuracy compared to the original model. Furthermore, the experimental results showed that an edge device performs network traffic classification far faster than a cloud-based architecture. Based on this, we believe that this provides quantitative evidence and shows that performing network traffic classification with an edge device is both feasible and efficient.

### B. Limitations

The major limitation of this study is the edge device we used. While the WIO Terminal has internet connectivity, it

does not support actively sniffing out packets itself. Because of this, only an approximation of an edge device performing network traffic classification could be used to collect results. Furthermore, the small amount of memory and space of edge devices limit the program size and how much traffic they can realistically capture and compute at once. Lastly, our study only performs known malware traffic classification, meaning that we are unsure about the prospects of classifying unknown traffic.

### C. Future Work

Because of the limitations we faced, we could only do a simulation of an edge-computing architecture instead of actually creating one. In the future, we would like to create the architecture itself and show its functionality and prowess in performing network traffic classification. Furthermore, we would like to research conducting network traffic classification on unknown traffic on the edge as it is important for systems like a network intrusion detection system. Lastly, given some promising results in moving network traffic classification to the edge, we would like to research and promote research about the feasibility of moving other standard cybersecurity procedures to the edge.

## VI. Acknowledgement

This research was supported by the National Science Foundation under award No. 2150248.

## References

[1] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 1, pp. 104–117, 2013. DOI: 10.1109/TPDS.2012.98.

[2] Y. Pan, X. Zhang, H. Jiang, and C. Li, "A network traffic classification method based on graph convolution and lstm," IEEE Access, vol. 9, pp. 158 261–158 272, 2021. DOI: 10.1109/ACCESS.2021.3128181.

[3] L. S. Vailshery. "Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030." (May 2022), [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (visited on 06/30/2022).

[4] J. Pan and Z. Yang, "Cybersecurity challenges and opportunities in the new "edge computing + iot" world," in Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks Network Function Virtualization, ser. SDN-NFV Sec'18, Tempe, AZ, USA: Association for Computing Machinery, 2018, pp. 29–32, ISBN: 9781450356350. DOI: 10.1145/3180465.3180470. [Online]. Available: https://doi.org/10.1145/3180465.3180470.

[5] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," IEEE Access, vol. 8, pp. 85 714–85 728, 2020. DOI: 10.1109/ACCESS.2020.2991734.

[6] Y. Y. Siang, M. R. Ahamd, and M. S. Z. Abidin, "Anomaly detection based on tiny machine learning: A review," Open International Journal of Informatics, vol. 9, no. Special Issue 2, pp. 67–78, Nov. 21, 2021.

[7] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in 2017 International Conference on Information Networking (ICOIN), Jan. 2017, pp. 712–717. DOI: 10.1109/ICOIN.2017.7899588.

[8] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," IEEE/ACM Trans. Netw., vol. 23, no. 4, pp. 1257–1270, 2015, ISSN: 1063-6692. DOI: 10.1109/TNET.2014.2320577. [Online]. Available: https://doi.org/10.1109/TNET.2014.2320577.

[9] Z. Fan and R. Liu, "Investigation of machine learning based network traffic classification," in 2017 International Symposium on Wireless Communication Systems (ISWCS), Aug. 2017, pp. 1–6. DOI: 10.1109/ISWCS.2017.8108090.

[10] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. K. Karn, and F. Abdessamia, "Network traffic classification techniques and comparative analysis using machine learning algorithms," in 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Oct. 2016, pp. 2451–2455. DOI: 10.1109/CompComm.2016.7925139.

[11] B. Yamansavascilar, M. A. Guvensan, A. G. Yavuz, and M. E. Karsligil, "Application identification via network traffic classification," in 2017 International Conference on Computing, Networking and Communications (ICNC), Jan. 2017, pp. 843–848. DOI: 10.1109/ICCNC.2017.7876241.

[12] S. Valenti, D. Rossi, A. Dainotti, A. Pescapè, A. Finamore, and M. Mellia, "Reviewing traffic classification," in Data Traffic Monitoring and Analysis, Springer, 2013, pp. 123–147.

[13] H.-K. Lim, J.-B. Kim, J.-S. Heo, K. Kim, Y.-G. Hong, and Y.-H. Han, "Packet-based network traffic classification using deep learning," in 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Feb. 2019, pp. 046–051. DOI: 10.1109/ICAIIC.2019.8669045.

[14] S. Rezaei and X. Liu, "Deep learning for encrypted traf-fic classification: An overview," IEEE Communications Magazine, vol. 57, no. 5, pp. 76–81, May 2019. DOI: 10.1109/MCOM.2019.1800819.

[15] K. Kim, J.-H. Lee, H.-K. Lim, S.-W. Oh, and Y.-H. Han, "Deep rnn-based network traffic classification scheme in edge computing system," Computer Science and Information Systems, vol. 19, no. 1, pp. 165–184, 2022. DOI: 10.2298/CSIS200424038K.

[16] Wio terminal user manual d51r, version 2, Seeed Studio, pp. 1–10. [Online]. Available: https://files.seeedstudio.com/wiki/Wio-Terminal/res/Wio-Terminal-User-Manual.pdf (visited on 06/12/2022).