Scalability of Neural Network Models for the Classification and Detection of Threats in Network Traffic

MSc Research Project
Cloud Computing

Erick Ayala Rodriguez
Student ID: 21176221

School of Computing
National College of Ireland

Supervisor: Punit Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Erick Ayala Rodriguez……………………………………………………………………………
Student ID:	21176221…………………………………………………………………………………….……
Programme:	MSc in Cloud Computing ………………… **Year:** 2023…….
Module:	Research Project ……………………………………………………….………
Supervisor:	Punit Gupta ……………………………………………………………………….………
Submission Due Date:	14/08/2023………………………………………………………………………………….………
Project Title:	Scalability of Neural Network Models for the Classification and Detection of Threats in Network Traffic…………………………………….………
Word Count:	9430………………………… **Page Count** 27 pages…………………………….…….

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Erick Ayala Rodriguez [illegible] ………………………………… ……………………………

Date: 12/08/2023………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Scalability of Neural Network Models for the Classification and Detection of Threats in Network Traffic

Erick Ayala Rodriguez
Student ID: 21176221

Abstract

This study evaluates the impact of parallel data processing implementing multiple GPUs and multiple core CPUs on neural network models for network traffic analysis. The UNSW-NB15 dataset will be used in experiments to test various machine learning models, including Convolutional Neural Networks (CNN), and Sequential Neural Networks (SNN). For reference, was implemented an additional model like Decision Tree Classifier (DTC) to have a comparison with traditional models. TensorFlow was selected as the main framework to work with due to the compatibility with Nvidia GPUs and the CUDA libraries, also the models were implemented using a local machine and then replicated the scenarios in an EC2 instance in AWS to test the scalability of the models incrementing the processing power with multiple GPUs and increasing the number of cores of the CPU. The results show a decrease in the time to train the models which can be helpful to develop new approaches in cybersecurity to detect and classify attacks while utilizing all the capabilities of the systems resources available and achieve lower training times and more accurate detections of network threats. The research aims to contribute to the development of innovative approaches to improve the accuracy and performance of neural network approaches for network traffic analysis.

1 Introduction

Cloud computing has become an essential technology for data storage and data processing, making it a primary target for cyber attackers (Eltaeib & Islam, 2021). Therefore, it is essential to develop effective techniques to detect and classify network traffic to ensure the security and availability of data and services. Machine learning (ML) has shown great potential for detecting and classifying network traffic (Tiwari & Jain, 2022). ML algorithms are capable of analyzing large amounts of data to identify potential threats in real-time, making them a valuable tool for intrusion detection.

Anomaly detection is a popular approach for network traffic analysis using machine learning algorithms. However, processing large datasets to train the models required a large amount of data which can take a long time to train the ML models and in the case of anomaly detection in network traffic is equally applicable. The use of graphic processing units (GPUs) can accelerate the performance of machine learning algorithms, particularly for deep learning models (Ilievski, Zdraveski, & Gusev, 2018).

Several studies have investigated the use of machine learning algorithms for network traffic analysis (Tiwari & Jain, 2022, Chkirbene et al., 2021, Salman et al, 2017, Divakar, Priyadarshini, & Mishra, 2020, Moustafa & Slay, 2015) and Ilievski, Zdraveski, & Gusev (2018) proved that the use of GPUs in machine learning can enhance performance. However, needs to be studied further the use of parallel data processing to test the scalability of the models in network threat detection and classification and calculated the limit of the scalability while adding more cores in a CPU and multiple GPUs to work in parallel. Therefore, this study aims to evaluate the impact of using multiple GPUs and multiple cores in a CPU for data processing in network traffic analysis using on-premise and cloud computing architectures and how it can scale for training large sets of data processed in parallel.

To achieve the research goals, were conducted experiments using the UNSW-NB15 dataset (Moustafa & Slay, 2015) to compare the training time of neural network models with GPUs and CPUs with multiple cores. The models tested will include Convolutional Neural Networks, Sequential Neural Networks, and the traditional Decision Tree Classifier to make comparisons. The performance of these models will be evaluated using metrics such as accuracy, precision, recall, and F1-score.

The experimental setup for the on-premise architecture involves a CPU with 6 cores and 12 threads, one Nvidia RTX 2060 GPU, CUDA library, NCCL library to be able to use strategies for data processing in parallel, Numpy, Pandas, Scikit, and TensorFlow framework in combination with Jupyter Notebook as the web-interactive computing platform. For the cloud architecture, was used an EC2 instance in Amazon Web Services (AWS) cloud. The experiments will be conducted using Jupyter Notebook running on a Linux operating system. The TensorFlow framework will be set up on the AWS EC2 instance type g3.8xlarge with 2 Nvidia Tesla M60 GPUs and a CPU with 32 virtual cores and 244 GB of RAM.

This research aims to contribute to the knowledge of cybersecurity and the effectiveness of ML, parallel data processing, and the scalability of the ML models in detecting and classifying threats in network traffic. The results of this research will provide valuable insights for cybersecurity providers and organizations using ML approaches to improve their network security and protect against potential threats by implementing just the necessary resources in computing power and avoiding oversizing of resources.

2 Related Work

The rapid growth of cloud computing has led to an increase in network security threats in cloud computing environments. To address this challenge, researchers have proposed the use of ML algorithms and GPU data processing to analyze network traffic to detect and classify potential threats (Tiwari & Jain, 2022; Chkirbene et al., 2021; Salman et al., 2017; Divakar, Priyadarshini, & Mishra, 2020; Pwint & Shwe, 2019; Moustafa & Slay, 2015; Hung & Wang, 2014). Notably, some of these studies have been reiterated, such as the work of Salman et al. (2017), underlining their significance in the field. Additionally, this review will discuss the potential benefits of parallel data processing and how it benefits the performance of the ML models.

2.1 Machine Learning Techniques for Network Traffic Analysis

The domain of computer network security remains a major concern in the IT industry, mainly because of the constant emergence of different attack methodologies. The studies by Tiwari & Jain (2022), Chkirbene et al. (2021), Salman et al. (2017), Divakar, Priyadarshini, & Mishra (2020), Pwint & Shwe (2019), Moustafa & Slay (2015), and Hung & Wang (2014) highlight various network security threats like DDoS attacks, botnet attacks, and malware infections. To detect and mitigate these threats, real-time network traffic analysis is essential, utilizing mechanisms that ensure swift detection combined with high detection accuracy. Ilievski, Zdraveski, & Gusev (2018) posited that employing machine learning and GPU data processing techniques can yield enhanced performance outcomes compared to traditional CPU-based approaches, especially when processing large data volumes in parallel.

Two additional investigations deep dives into this area of research were conducted by G. A. Haidar and C. Boustany in 2015 and the exploratory study by A. Aljohani and A. Bushnag in 2021. Haidar and Boustany capture the growing gravitas of anomaly-centric intrusion detection methods. They implemented neural networks to equip the detection systems with a suite of advanced threat reconnaissance and mitigation techniques.

Studies like Tiwari & Jain (2022), Chkirbene et al. (2021), Salman et al. (2017), Divakar, Priyadarshini, & Mishra (2020), Pwint & Shwe (2019), Moustafa & Slay (2015), and Hung & Wang (2014) defend the use of machine learning techniques in network traffic analysis. Also, Chkirbene et al. (2021) proposed a multi-stage approach that used unsupervised clustering alongside deep learning models to discern and categorize network traffic. Tiwari & Jain (2022) implemented a convolutional neural network (CNN) to categorize incoming traffic, distinguishing it as either regular or anomalous. Salman et al. (2017) suggested a machine learning-centric methodology for the detection and categorization of anomalies in multi-cloud settings. All these studies highlight the efficacy of machine learning techniques in the detection and classification of network traffic and spotlight the prospective advantages these methods hold for augmenting cloud security.

The investigations demonstrate the effectiveness of these techniques for detecting and classifying network traffic and show the potential benefits. However, the papers also highlight some challenges, such as the need for large datasets to train the models and the potential for high false positive rates as Chkirbene et al. (2021), Salman et al. (2017), and Pwint & Shwe, 2019 mention. Additionally, the investigations from Tiwari & Jain, (2022), Salman et al. (2017), and Hung & Wang, 2014 provide insights into potential areas for further research, such as the use of reinforcement learning techniques for network traffic analysis.

2.2 Parallel data processing

The speed of the detection of threats is vital for the security of the information to prevent leakage of it. The use of high-capacity resources like CPUs with a high amount of cores or GPU for data processing is crucial to accelerate training in ML algorithms, Deep Learning (DL) has rapidly evolved, with a significant focus on the efficiency of computational frameworks. A study by Chien et al. (2018) underscored the role of data management in how it is read and stored in TensorFlow. Ilievski, Zdraveski, & Gusev. (2018) highlights the role of CUDA in accelerating machine learning algorithms using GPUs. In a second investigation, Chien et al. (2019) tested the TensorFlow framework potential and proved its utility in High-Performance Computing (HPC). To validate this, they constructed four standard HPC tests executed on expansive computers or "supercomputers" using TensorFlow.

Sattar and Anfuzzaman (2020) pointed out the Sparse DNN challenge by employing data parallelism using GPUs, leveraging Python in TensorFlow. Their work achieved up to 4.7 times faster speeds than conventional MATLAB implementations and their GPU-based solution showcased a thrice-fold speed increase compared to its CPU counterpart.

Gupta, Maity, Das, and Wandhekar (2021) investigated the best software distribution combination for TensorFlow, focusing on optimizing CPU usage in High-Performance Computing (HPC) clusters. In their experiments, the authors noted that Intel's distribution for TensorFlow has superior performance across different CPUs and Python distributions. Their findings offer up to a sevenfold performance enhancement on specific CPU systems, and when distributed training techniques were employed, they noticed an additional 1.4-fold performance improvement.

Divakar, Priyadarshini, & Mishra. (2020) demonstrates how Numba can be used to accelerate Python functions on GPUs, and highlights its potential benefits for accelerating machine learning algorithms in network traffic analysis.

Ramirez-Gargallo, Garcia-Gasulla, and Mantovani (2019) explored TensorFlow's performance in state-of-the-art HPC clusters. Their study identified the most efficient hardware/software configurations to enhance TensorFlow's performance in HPC settings.

In terms of distributed training, Quang-Hung, Doan, and Thoai (2020) emphasized the role of High-Performance Computing (HPC) in DL training. Given the extensive data, DL models must iterate over, training without HPC could span weeks or months. Their research spotlighted distributed training as a solution to expedite DL's training process. Evaluating TensorFlow 2.2, they presented benchmarking results on distributed training for datasets like MNIST and Cifar-10 and other distributed training strategies in clustered GPU and/or TPU environments.

Campos, Sastre, Yagües, Torres, and Giró-I-Nieto (2017) emphasized the importance of parallelizing the training of a deep neural network on a distributed GPU cluster. They utilized TensorFlow on Barcelona Supercomputing Center’s GPU cluster, focusing on the convolutional neural network’s performance for classifying adjective-noun pair and the research showed a reduction of training times from a whopping 106 hours to 16 hours.

Bagby, Rao, and Sim (2018) shifted the spotlight to the Recurrent Neural Network Transducer (RNN-T) and they efficiently implemented the RNN-T forward-backward and Viterbi algorithms in TensorFlow. The emphasis was on maximizing parallel computation by adjusting batch sizes, and analyzing performance across various hardware, including GPU and TPU architectures. Interestingly, their results showed the TPU performance was about twice as fast as the GPU setup, with a batch size of 32.

Malik et al. (2018) emphasized the challenges and bottlenecks associated with training high-accuracy DNNs, especially when relying on single GPU setups and they noted that training the GoogleNet with the ImageNet dataset on a single Nvidia K20 GPU takes 25 days proving the necessity of implementing Distributed Deep Neural Networks (DDNNs) to accelerate training times. The authors highlighted the difficulties in modifying deep learning frameworks or libraries to facilitate this inter-node communication in TensorFlow so they explored the Horovod framework, which promises ease of use for distributed learning across platforms like TensorFlow, PyTorch, and Keras showing that the Horovod framework offers almost linear throughput scalability up to 256 GPUs (Malik, Lu, Wang, Lin, & Yoo, 2018).

Jain et al. (2019), on the other hand, offered a fresh perspective by shifting the focus from GPUs to CPU-based DNN training providing a detailed test of the performance of DNNs like ResNet(s) and Inception variants on several CPU architectures and GPUs achieving significant speedups on multiple nodes, especially when used in conjunction with tools like TensorFlow with Horovod and the MVAPICH2 MPI library.

The investigations demonstrate the effectiveness of ML in parallel distribution of workload for data processing and training. The authors highlight the potential benefits of these techniques to accelerate data processing. However, further research is needed to address the challenges identified and to explore the use of emerging techniques such as the combination of scalable techniques to process a large amount of data in less time and test different machine learning models and combinations of algorithms to increase accuracy when network threat detection is the objective.

The next section will detail the approach of this research addressing the challenges related to the time to train Neural Networks algorithms in the detection and classification of network threats using multiple core CPUs & GPUs to test the scalability of the algorithms to process the data training in parallel, and CUDA libraries, the framework TensorFlow, and how the scalability work on each ML models rested.

3 Research Methodology

The aim of this thesis is to evaluate the impact of using multiple GPUs and multiple core CPUs in training Neural Network algorithms and how the speed of training can affect the accuracy of the models for network traffic analysis. To achieve this, two research questions have been identified:

- What are the machine learning models that process data in less time with parallel data processing in the field of network threat detection and classification?
- How do multiple GPUs and multicore CPUs scale in training times of ML algorithms for the detection and classification of threats in network traffic analysis?
- How can parallel data processing improve or affect the accuracy and performance of ML approaches for detecting and classifying threats in network traffic?

The proposed solution involves conducting experiments using the UNSW-NB15 dataset (Moustafa & Slay, 2015) to compare the performance of the training process with CPUs & GPUs and how it scales using multiple GPUs & Cores in parallel. The ML models to be tested include Convolutional Neural Networks (CNN), Sequential Neural Networks (SNN), and Decision Tree Classifier (DTC) because due to the nature of how they operate can be utilized to process data in parallel taking advantage of the parallel distribution of workload during the training process. The performance of these models will be evaluated using metrics such as time to train, accuracy, precision, recall, and F1 score.

After the development of the code, it is going to be tested the scalability of the models using multiple GPUs and multiple cores in the CPU in parallel using Tensorflow to orchestrate the workload across the workers.

Next is going to detail the hardware, software, architecture proposed, and methodology considered in this research.

3.1 Hardware and software approach

The experimental setup will involve a CPU with 6 cores and 12 threads, one GPU, CUDA library[1], NumPy, Pandas, Scikit, TensorFlow framework, and Windows 11 Home as the operating system for the on-premise architecture. For the cloud architecture, the experiments will be conducted using the same libraries but with a Linux operating system and two GPUs to test the scalability of the ML models. Both scenarios will use Jupyter Notebook to run the code.

3.2 Local architecture

As mentioned in section 3.1, the first test is going to be set in a local machine with 32 GB of RAM, one CPU AMD Ryzen 5 4600H[2], and one Nvidia RTX 2060[3] GPU with 6 GB of VRAM.

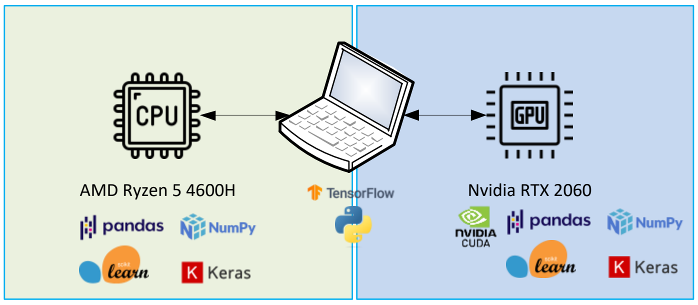


Diagram 1: Local architecture.

Diagram 1 shows the tow first scenarios to be tested, each scenario is going to follow the next logic:

a) CPU test:
   a. Using Python in combination with the Tensorflow framework and the libraries Numpy, Pandas, Scikit Learn, and Keras is going to be created the code to process the dataset
   b. Then as a traditional approach, is going to be used the AMD processor to process the workload

b) GPU test:
   a. Using Python in combination with the Tensorflow framework and the libraries Numpy, Pandas, Scikit Learn, Keras, and Nvidia CUDA is going to be created the code to process the dataset

[1] https://developer.nvidia.com/cudnn
[2] https://www.amd.com/en/product/9086
[3] https://www.nvidia.com/en-gb/geforce/gaming-laptops/compare-20-series/

b. Then instead of using the CPU to process the workload, the Nvidia GPU is going to be used to process the workload due to the use of the CUDA library.

Next is going to be described the algorithms and procedures of each model considered for the research and at the end of this section will be explained the architecture and procedures to implement the models for scalability using parallel data processing in the training process.

3.3 Cloud architecture

To implement the models in the AWS cloud is going to be used an Amazon EC2 instance type g3.8xlarge[4] with up to 2 Nvidia Tesla M60[5] GPUs with NCCL library[6] to make use of parallel distribution of the workload between the two GPUs during the training time.

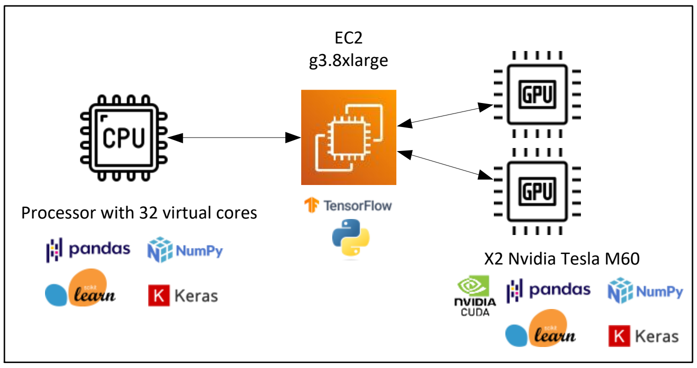


Diagram 2: Local architecture.

Diagram 2 shows the cloud architecture tests which are going to have two separate tests, one using the CPU and the second test using GPU. The difference between the architectures is that the EC2 instance created in the AWS cloud has more resources available to test the scalability of the ML models using more cores on the CPU side and multiple GPUs on the other in addition to the CUDA and NCCL libraries with the same purpose as mentioned.

Also is important to mention that the EC2 instance does not require more than one instance because the purpose is to test the scalability of the model using parallel workloads and the instance has enough resources to do the necessary tests.

3.4 Testing methodology

The scalability measurement of the ML models is going to follow the next steps:

a) Will be Measured the time to train the ML models using single and multiple GPUs.

b) Will be measured the time to train the ML models using single and multithread configurations up to 32 threads to make a match with the hardware available.

[4] https://www.amazonaws.cn/en/ec2/instance-types/
[5] https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/solutions/resources/documents1/nvidia-m60-datasheet.pdf
[6] https://developer.nvidia.com/nccl

c) Will be compared the on-premise and the cloud architecture using 12 threads (matching the number of cores of the CPU available in the on-premise) to compare how the architecture and the frequency of the processor affect the speed in the training process.
d) Also will be compared the use of a single GPU in bout architectures to see how the VRAM available, the architecture and the resources of each GPU affects the training speed.

3.5 Framework and algorithms

An important part of this research is to build an effective ML model for network threat detection and classification. To do this was implemented the ML models like CNN, SNN, and DTC. The aspect of detection and classification of network threats will be helpful to identify patterns in the network packets from the dataset.

To detect the threats was used a binary classification to represent if the network traffic represents a threat or not. The classification of threats was implemented as a multi-class classification aimed to classify network traffic into specific attack categories.

To make use of all the hardware available in the systems and to reduce the time to train the ML models, it is used the framework TensorFlow in combination with libraries like Pandas, Numpy, Scikit Learn, Keras, and CUDA (CUDA libraries are used only when the training is done by the GPUs). The configuration of the framework and the ML models are described in Section 4 “Design specification”.

3.6 Logic of the ML models

To implement the ML models, the steps as shown in Diagram 2 will be executed for each model:

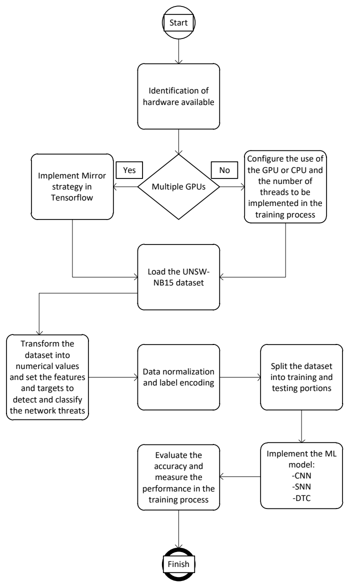

Start
Identification of hardware available
Multiple GPUs
Yes
No
Implement Mirror strategy in Tensorflow
Configure the use of the GPU or CPU and the number of threads to be implemented in the training process
Load the UNSW-NB15 dataset
Transform the dataset into numerical values and set the features and targets to detect and classify the network threats
Data normalization and label encoding
Split the dataset into training and testing portions
Implement the ML model: -CNN -SNN -DTC
Evaluate the accuracy and measure the performance in the training process
Finish

Diagram 2: Logic of data preparation and processing.

a) Identification of hardware using the “os” module in Python to validate the visibility of the GPUs and CPUs from TensorFlow.

b) Implement the Mirrored Strategy in TensorFlow to make use of the GPUs available in parallel when the test is selected for this reason or make use of one GPU or CPU core based on the threads configured in the variables.
c) Load the UNSW-NB15 and merge the training and testing sets to have a larger dataset.
d) Transform the dataset to use numerical values instead of characters or strings using label encoders and set the features and targets in the dataset to detect and classify network threats.
e) Perform data normalization because the dataset has different ranges in the values and use label encoding for the detection of network threats (binary classification) and classify the network threats (multiclass classification)
f) Split the dataset into training (80%) and testing (20%)
g) Implement the training and testing of each ML model.
h) Measure the training time using the different approaches and ML models, and evaluate the accuracy, precision, recall, and F1 score.

4 Design Specification

This section will be described how was configured TensorFlow and the ML models to use the GPUs in parallel and multiple cores using multi-threading specifications in the code. It is important to mention that some parameters need to be set up in the code of the ML models to have the capability to switch between a single GPU to multiple GPUs or the use of one or more threads to control the number of cores of the CPU available in the system.

4.1 Tensorflow

To accelerate training was implemented variables to control de number of threads when the CPU is the device selected to do the data processing and a variable to control the number of GPUs when this device is selected for data processing. The purpose of using GPUs and CPUs with multithread configurations is to perform high-speed numerical computations in parallel and measure how it reduces the training time in the ML models each device.

To handle large datasets and model parallelism, TensorFlow was configured using distributed training like "tf.distribute.MirroredStrategy[7]" to enable data and model parallelism in all the GPUs and multithreading environmental variables to control and make use of all the cores in the CPUs available for the tests. It is worth to mention that this strategy is not necessary when the CPU is selected or the system only have one GPU.

4.2 Convolutional Neural Network architecture

For the use of CNN it was implemented two different approaches, one for the detection of network threats and another one for the classification of the threat to evaluate the neural network, in this section will be described the logic behind the ML model configuration.

[7] https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy

4.2.1 Detection of network threats CNN (binary classification).

The ML model architecture has three 1D Convolutional layers with 32, 128, and 256 filters and a kernel size of 3 to capture temporal patterns from the sequential data of network attributes. Rectified Linear Unit (ReLU) was used to activate the function for the Convolutional layer, introducing non-linearity and allowing the model to learn complex representations in combination with a Max Pooling layer with a pool size of 2 to reduce spatial dimensions and enhance the model's ability to generalize. Two Dense layers with 512 and 256 units with ReLU activation was used to learn non-linear combinations of features extracted by the previous layers and to prevent overfitting a Dropout layer with a rate of 0.5 was inserted. Lastly, a final layer has a single unit with a Sigmoid activation function, producing a probability value between 0 and 1 (binary classification of threats to validate the likelihood of an attack in the network). To represent the model, Diagram 3 shows the flow of it.

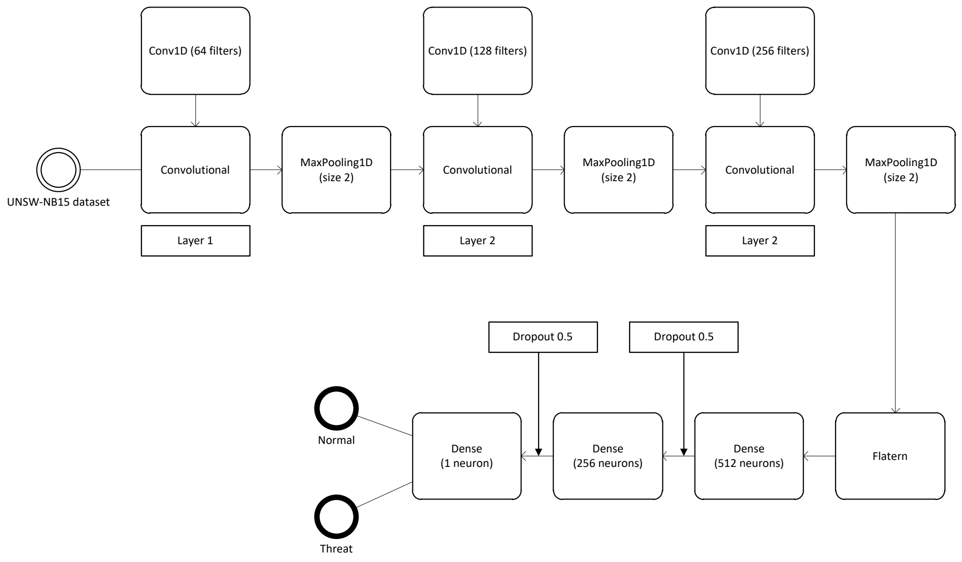


Diagram 3: CNN binary.

The training for the binary classification implemented the Binary Cross-Entropy as the loss function, to measure the difference between predicted probabilities and true binary labels, in combination with the task of the binary classification.

4.2.2 Classification of network threats CNN (multi-class classification).

The classification of network threats architecture has one 1D Convolutional layer with 32 filter, one Dense layer with 1024 fully connected neurons. It uses a Softmax activation function to produce a probability distribution over the multiple classes from the dataset. Diagram 4 shows the flow of it.

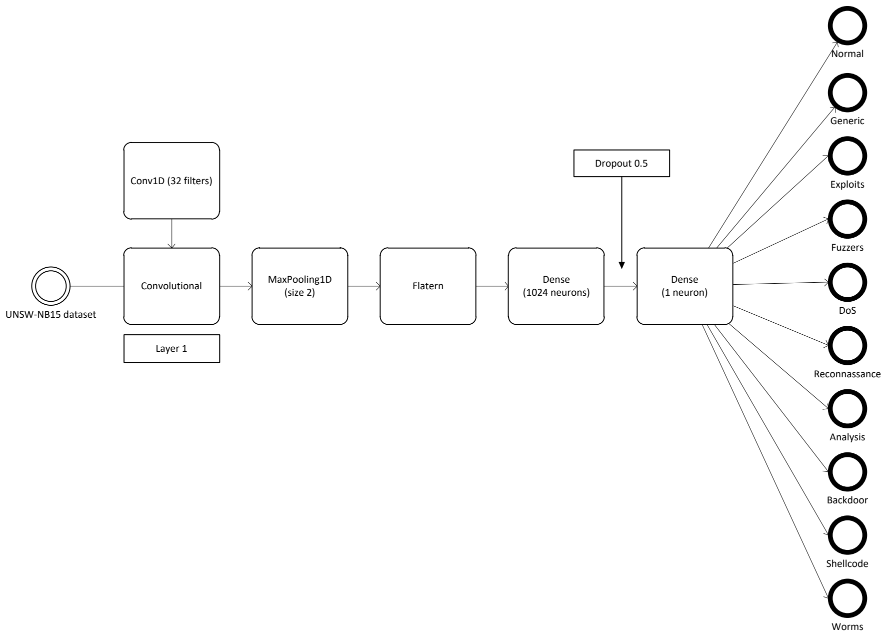


Diagram 4: CNN muti-class.

4.3 Sequential Neural Network

SNN was configured to test its reliability in the detection and classification of network threats because the logic behind it is sequential as the network packets are transmitted. The configuration for each classification task is described next.

4.3.1 Detection of network threats SNN (binary classification).

The SNN neural network was implemented in combination with Keras library for binary classification tasks and has three sequential layers added to the model container. The first layer used a dense layer with 64 units and the activation function ReLU and the second layer uses a dense layer with 32 also with the activation function ReLU and finally, the third layer uses a dense layer with a single unit and the "sigmoid" activation function which is used because it is designed for the binary classification outputs needed for the detection of threats (it is a threat or not or 1||0 respectively).

The SNN ML model is compiled using the "Adam" optimizer in combination with the loss function "binary cross-entropy" because the target variable is binary.

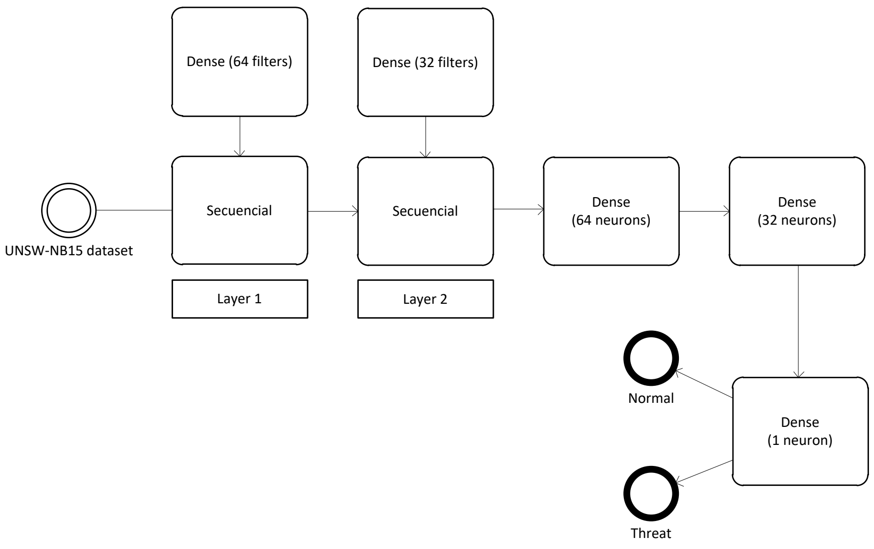


Diagram 5: SNN binary class.

4.3.2 Classification of network threats SNN (multi-class classification).

For the classification of network threats using SNN the ML model was configured similarly to the SSN for the binary classification. The difference is in the activation function which is "Softmax" and the loss function "categorical cross-entropy" because are commonly used for multi-class classification tasks as needed to classify the threats.

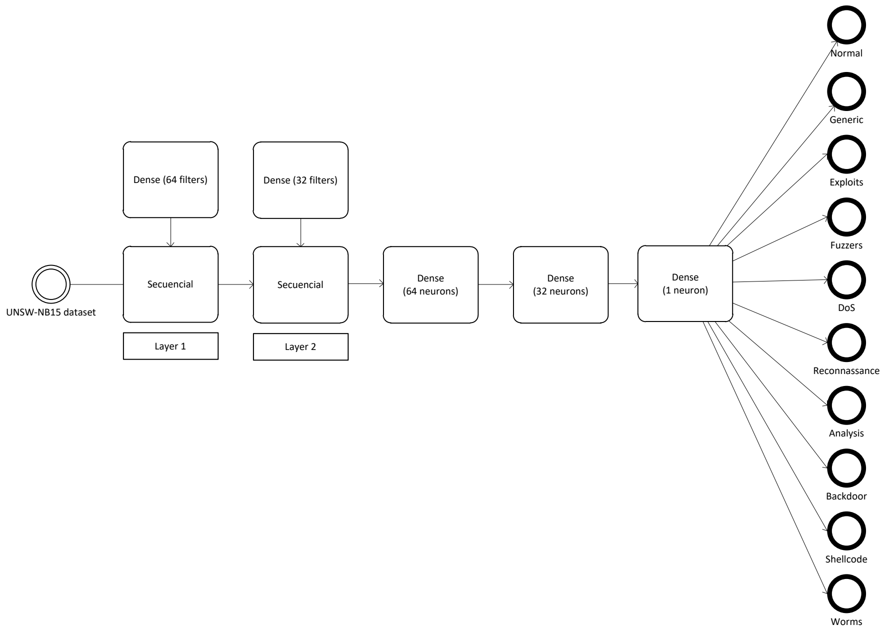


Diagram 6: SNN muti-class.

4.4 Decision Tree Classifier

The DTC ML model implementation was considered to have a different approach apart from neural networks and see how it performs and scales while using single and multiple GPUs and CPUs with one or more cores. Same as the neural network models, DTC was also configured to detect and classify threats and it will be described in the next section.

4.4.1 Detection and classification of network threats DTC (binary and multi-class classification).

For the detection and classification using DTC was implemented the parameter "random state" to ensure reproducibility, then is used the "fit" model to train the DTC using the pre-processed and encoded features from the training dataset and their labels. The same logic in the model was implemented for the detection and classification of network threats only changing the target variable.

DTC works with feature extraction, it was considered for the tests because it can be used for network threat detection by leveraging its ability to make binary or multiclass predictions based on the features extracted from network traffic data.

5 Implementation

The implementation of the ML models is explained in the present section. It will be explained how the architecture mentioned in the present research (sections 3.2 and 3.3) was utilized for the implementation of the ML models and the information obtained from the dataset to proceed with the data cleaning, normalization, standardization, training, and evaluation of the dataset.

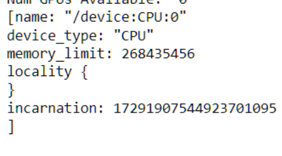
5.1 Distribution of workload for parallel processing

For the implementation of parallel processing was necessary to use libraries in Python to validate that the framework is capable to reach the hardware available, select the hardware needed for the test (GPU or CPU), and configure the required strategy in TensorFlow. The following sections will explain the implementation done to set up the variables for the tests done.

5.1.1 Using CPU or GPU for data processing

For the selection of CPU, was necessary to disable the use of the GPU because TensorFlow takes it by default if the drivers and libraries are properly installed. To do it, is necessary to import the "os" python library and set the environmental variable of the "CUDA_VISIBLE_DEVICES[8]" in "-1" to disable the use of GPUs.

Then is necessary to use the environmental variable "TF_NUM_INTEROP_THREADS[9]" and "TF_NUM_INTRAOP_THREADS" to set up the number of threads to process the data. Figure 1 and Figure 2 shows the output after using GPU or CPU in the On-premise architecture and Figure 3 and 4 shows the output from the Cloud architecture.



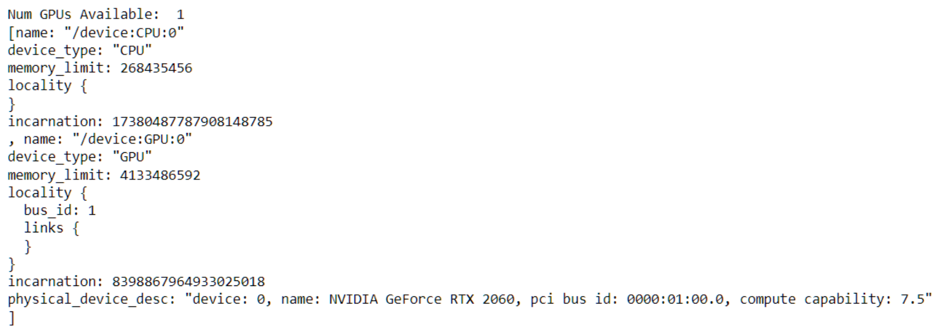
```
Num GPUs Available:  0
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 17291907544923701095
]
```

Figure 2: On-premise architecture CPU output.

[8] https://developer.nvidia.com/blog/cuda-pro-tip-control-gpu-visibility-cuda_visible_devices/

[9] https://www.tensorflow.org/api_docs/python/tf/config/threading/set_inter_op_parallelism_threads

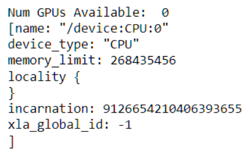
As shown in Figure 1, the output shows that the system can not see the GPU, and Figure 2 shows that 1 GPU is available and provide more detail about it which in the On-premise architecture is an Nvidia RTX 2060.



```
Num GPUs Available:  1
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 17380487787908148785
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 4133486592
locality {
  bus_id: 1
  links {
  }
}
incarnation: 8398867964933025018
physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 2060, pci bus id: 0000:01:00.0, compute capability: 7.5"
]
```

Figure 2: On-premise architecture GPU output.

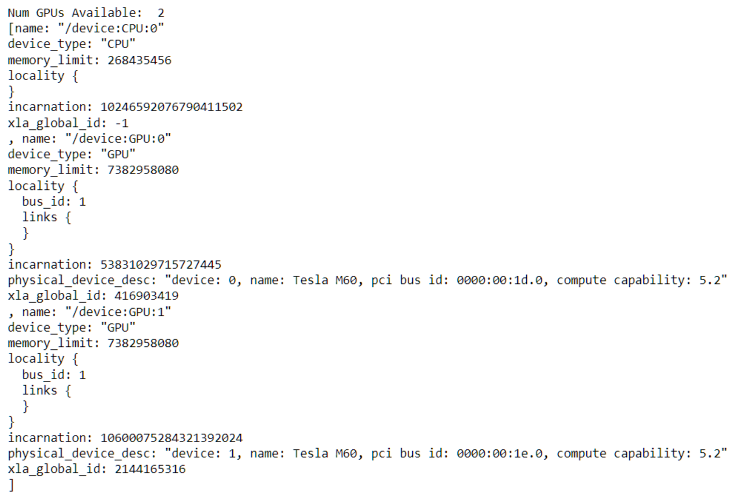
For the Cloud architecture, the output is shown in Figures 3 and 4.



```
Num GPUs Available:  0
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 9126654210406393655
xla_global_id: -1
]
```

Figure 3: Cloud architecture CPU output.

Figure one equally to the on-premise architecture shows no GPU availability for the system and Figure 4 shows the detail of the GPUs available which are two Nvidia Testa M60.

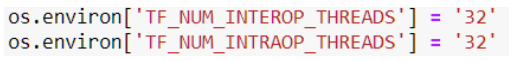


```
Num GPUs Available:  2
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 10246592076790411502
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 7382958080
locality {
  bus_id: 1
  links {
  }
}
incarnation: 53831029715727445
physical_device_desc: "device: 0, name: Tesla M60, pci bus id: 0000:00:1d.0, compute capability: 5.2"
xla_global_id: 416903419
, name: "/device:GPU:1"
device_type: "GPU"
memory_limit: 7382958080
locality {
  bus_id: 1
  links {
  }
}
incarnation: 10600075284321392024
physical_device_desc: "device: 1, name: Tesla M60, pci bus id: 0000:00:1e.0, compute capability: 5.2"
xla_global_id: 2144165316
]
```

Figure 4: Cloud architecture GPU output.

5.1.2 Using threads or distributed mirrored strategy

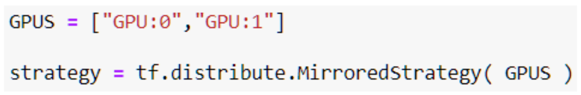
After the selection of which hardware is required for the data processing, it is necessary to configure the number of threads which the data processing will be split into when the CPU is selected. The definition of threads can be done also for GPUs but it will not make any difference because CUDA libraries will handle the data in parallel by default to make use of all the resources available in the GPU, Figure 5 shows how was implemented the environment variables set for 32 threads.



```
os.environ['TF_NUM_INTEROP_THREADS'] = '32'
os.environ['TF_NUM_INTRAOP_THREADS'] = '32'
```

Figure 5: Implementing threads for CPU.

If GPU is selected as the hardware to process the data, is necessary to eliminate the code to set the number of threads and implement the distribute mirrored strategy in TensorFlow. Figure 6 shows how was implemented.



```
GPUS = ["GPU:0","GPU:1"]

strategy = tf.distribute.MirroredStrategy( GPUS )
```

Figure 6: Implementing distribute mirrored strategy for GPU.

To make use of multiple GPUs in parallel implemented it is important to get the device name and ID of the devices obtained from the output explained in Section 5.1.1, then the name and ID are used in the "strategy" variable as an argument to distribute the workload across the devices listed. After the definition of the strategy, was implemented in the creation of the ML model, compilation, and training process of the ML models.

5.2 Dataset analysis and performance measurement

To understand the dataset distribution, was necessary to obtain graphs to get visualize the number of normal network traffic, the abnormal network traffic or threats, and how many classes of threats are in the dataset.

Figure 7 shows the count of network traffic having the counter on the "label" column of the dataset which represents the target variable to detect a network threat. The target variable is binary and the label "0" represents normal network traffic and "1" is a network threat.

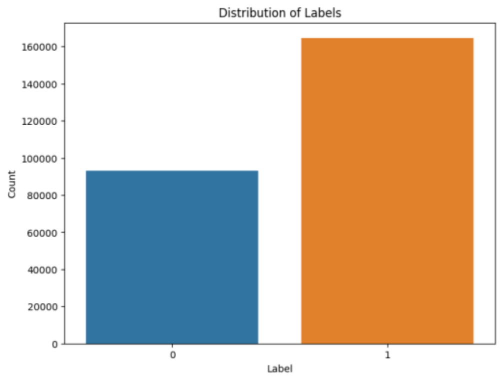


Figure 7: Binary classification count.

Figure 8 shows how the labels are classified by the type of network traffic, the normal network traffic remains the same and the threat network traffic is split into a multi-class classification according to the type of network attack showing 9 classes of network attacks.

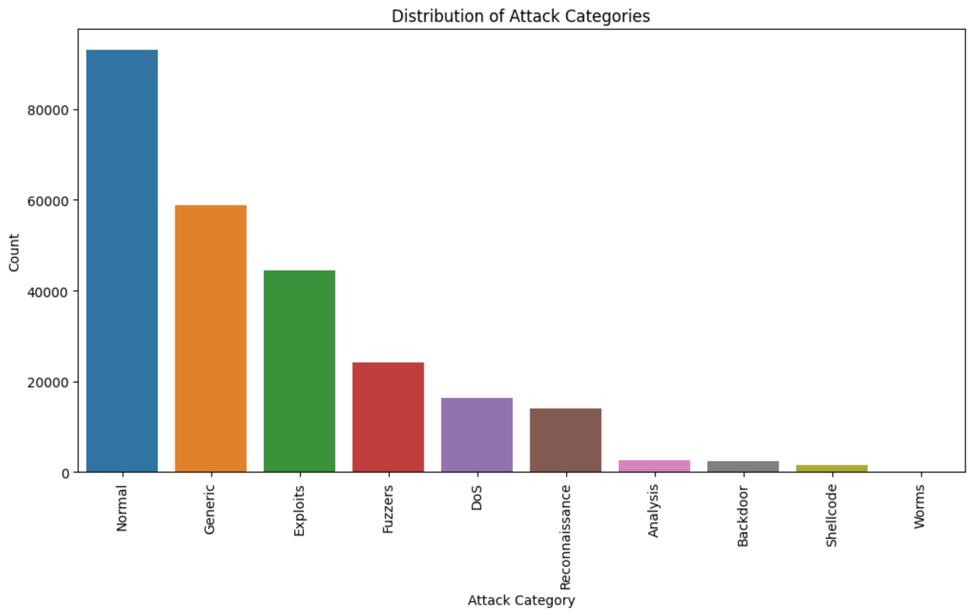


Figure 8: Multi-class count.

To measure the performance was implemented a time function measured the time from the training phase until the models finishes the training process. The results obtained in the testing phase are shown in the next section.

6 Evaluation

The tests executed have the purpose of analyzing the accuracy, precision, recall, and F1 score of each ML model in the detection and classification of network threats and at the same time evaluating the performance of the models using parallel data processing in the training process using single and multiple GPUs and muti-thread configurations to control the number of core in the CPU to be used.

This test aims to provide valuable information for future improvements in the performance of neural network models in the detection and classification of network threats using parallel data processing to reduce the training time and be aware of resource utilization to properly size the systems to be used to train the models. The next sections are going to describe the scenarios tested.

6.1 Case study 1: On-premise architecture

The first set of tests was performed using the on-premise architecture and is divided into two sections, one section is to test the model's scalability for the detection of network threats, and the second is for the classification of them.

6.1.1 Experiment 1: Detection of network threats (binary classification)

The first set of tests was performed in the on-premise architecture which has 1 CPU with 12 core and 1 Nvidia RTX 2060 GPU. The ML models were configured as mentioned in Section 5.1.1 to set the device to process the data. The results are shown in the next tables.

Table 1: Ryzen 4600H x1 Core x1 Thread Detection of threats (Binary classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	233.2915	0.9721	0.9782	0.9791	0.9772
Decision Tree Classifier	0.1810	1	1	1	1
SNN	27.7485	0.999	1	1	1

Table 2: Ryzen 4600H x12 vCore x12 Thread Detection of threats (Binary classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	191.8092	0.9733	0.9789	0.9789	0.9789
Decision Tree Classifier	0.1831	1	1	1	1
SNN	21.4814	0.9996	1	1	1

Table 3: x1 Nvidia RTX 2060 mobile Detection of threats (Binary classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	97.9310	0.9680	0.9746	0.9638	0.9856
Decision Tree Classifier	0.1762	1	1	1	1
SNN	87.2675	1	1	1	1

As shown in the results of Tables 1, 2, and 3 DTC has very good efficiency using single and multiple threads configurations as well as GPU, with low training times in every scenario. CNN is the only model that appears to reduce the time to train using the GPU and multi-threading. SNN shows improvements in training time using multi-threading but are marginal and when the GPU is used the training time is slower than using the CPU.

In terms of accuracy, precision, recall, and F1 score, the CNN model shows high values across different hardware. In contrast, models like the Decision Tree Classifier, and SNN all demonstrate perfect metrics in the testing section but these results show that this can be due to overfitting and it needs to be performed a deeper analysis to validate the results.

Based on the results, CNN performance in the training times increased by 17.77% when transitioning from 1 thread to 12 threads and 58.03% when moving to a GPU showing that CNN is the model which benefits the most from more resources process data in parallel. Other models like SNN shows also an improvement of 22.58% in the training times by increasing the number of threads and cores in the CPU but the training time using GPUs is higher because the configuration of the model could not be complex enough and fewer threads are needed to perform the data training. The scalability of the DTC is negligible because the training times are less than one second in all the scenarios and the time difference can be in the margin of error.

6.1.2 Experiment 2: Classification of network threats (multi-class classification)

For the classification of network threats in the on-premise architecture the results are shown in the next tables.

Table 4: Ryzen 4600H x1 Core x1 Thread Classification of threats (Multi-class classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	115.5832	0.8372	0.8179	0.8372	0.8260
Decision Tree Classifier	2.4114	0.7941	0.5196	0.5889	0.5312
SNN	26.6075	0.7911	0.78	0.79	0.77

Table 5: Ryzen 4600H x12 Core x12 Thread Classification of threats (Multi-class classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	96.9567	0.8339	0.8144	0.8339	0.8199
Decision Tree Classifier	2.4395	0.7945	0.5202	0.5983	0.5351
SNN	21.5749	0.7801	0.77	0.78	0.79

Table 6: x1 Nvidia RTX 2060 mobile Classification of threats (Multi-class classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	55.5871	0.8360	0.8238	0.8360	0.8261
Decision Tree Classifier	2.4078	0.7941	0.5480	0.5531	0.5298
SNN	75.3286	0.7572	0.74	0.76	0.76

As shown in Tables 4, 5, and 6 CNN benefits again from multithreading having an increase in performance of 16.11% using 12 threads and cores in comparison to 1 thread and 1 core, for the GPU usage the acceleration in comparison to 1 thread and 1 core the performance in the training time increase by 51.90% in the classification of network threats. The DTC is efficient in terms of training time across all configurations but it does not show any improvement in the training time increasing the number of cores or the use of GPU capabilities and lags in the metrics in comparison to CNN and SNN. The SNN has an increase of 18.92% in the training time performance using multithreading with CPU configurations but same as the binary tests, it can be not complex enough to take advantage of the GPU given the increased training time and decreased accuracy.

6.2 Case study 2: Cloud architecture

Similarly, to the test presented in Section 6.1, the tests using the cloud architecture are divided into the detection and classification of network threats, these tests show scenarios where not only the number of cores in a CPU or the number of GPUs are important, but also the architecture of the devices matters in having good or bad results even when the same code and the same amount of threads are used.

6.2.1 Experiment 1: Detection of network threats (binary classification)

The cloud architecture has 1 CPU with 32 virtual cores and 2 Nvidia Tesla M60 GPUs. The ML models were configured as mentioned in Section 5.1.1 to set the device to process the data. The results are shown in the next tables.

Table 7: x1 vCore x1 Thread Detection of threats (Binary classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	209.0336	0.9694	0.9762	0.9833	0.9692
Decision Tree Classifier	0.1906	1	1	1	1
SNN	35.7686	0.9995	1	1	1

Table 8: x32 vCore x32 Thread Detection of threats (Binary classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	142.0379	0.9717	0.9776	0.9688	0.9865
Decision Tree Classifier	0.1907	1	1	1	1
SNN	35.1155	0.9996	1	1	1

Table 9: x1 Nvidia Tesla M60 Detection of threats (Binary classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	97.6922	0.9716	0.9776	0.9712	0.9841
Decision Tree Classifier	0.1930	1	1	1	1
SNN	101.6916	0.9999	1	1	1

Table 10: x2 Nvidia Tesla M60 Detection of threats (Binary classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	123.2225	0.9720	0.9782	0.9826	0.9737
Decision Tree Classifier	0.1983	1	1	1	1
SNN	131.0704	0.9999	1	1	1

The metrics like accuracy, precision, recall, and f1 score remain the same in comparison to the on-premise architecture and this was expected because the only aspect changed is the hardware so the values that should be changing are only the training times.

The results show that the training time with CNN was improved by 32.05% using 32 threads and 32 cores in comparison to 1 thread and 1 core, this demonstrates that the model still scales well in other hardware and also shows that the CPU architecture has an impact in the training times because the cloud CPU have better performance in single-core and single thread configuration leaving the performance to the hardware capabilities, also CNN shows an improvement in the training times using one GPU having an increase of 53.26% over the single thread configuration but on the performance in the training times is higher using two GPUs in comparison to one. This can be indicative that the workload is not big enough to make use of all the resources available adding more time because TensorFlow has to coordinate the workload across more devices.

The training time with DTC remains under one second and the times are so low that is impossible to notice any improvement by adding more cores or using multi GPUs.

Whit SNN the training time remains stable when it is changed from 1 thread to 32 threads showing no improvement with the use of multithreading in this scenario. This is interesting because the tests done in the om-premise architecture from Section 6.1.2 shows a small improvement but is still there when using multithreading configurations, this can represent the bigger capabilities of the CPU in the EC2 instance that proves that the model and workload are not complex enough to make use of all the resources. When the tests are done on a GPU, there is an increase in the training time, proving again that the workload is small for the hardware capabilities and the coordination of more devices only adds more time in the training process. Moving from a single thread to a single GPU the results show an increase to 101.69 seconds and on 2 GPUs, the training time increased even more to 131.07 seconds, similar to the CNN, suggesting that the SNN might not be optimized for GPU utilization, or there are overheads.

6.2.2 Experiment 2: Classification of network threats (multi-class classification)

The last section of tests is similar to the tests done in Section 6.1.2 having the target variable to classify the network threads.

Table 11: x1 vCore x1 Thread Classification of threats (Multi-class classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	108.2400	0.8351	0.8132	0.8351	0.8217
Decision Tree Classifier	2.6518	0.7941	0.5298	0.5531	0.5480
SNN	35.3595	0.7824	0.76	0.78	0.77

Table 12: x32 vCore x32 Thread Classification of threats (Multi-class classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	77.4408	0.8388	0.8269	0.8388	0.8294
Decision Tree Classifier	2.6559	0.7941	0.5298	0.5531	0.5480
SNN	33.1640	0.6745	0.66	0.67	0.7

Table 13: x1 Nvidia Tesla M60 Classification of threats (Multi-class classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	66.6308	0.8383	0.8239	0.8383	0.8296
Decision Tree Classifier	2.6578	0.7941	0.5298	0.5531	0.5480
SNN	95.1097	0.7719	0.75	0.77	0.76

Table 14: x2 Nvidia Tesla M60 Classification of threats (Multi-class classification)

ML model	Training time (s)	Accuracy	F1 score	Recall	Precision
CNN	95.2998	0.8332	0.8092	0.8332	0.8273
Decision Tree Classifier	2.6484	0.7941	0.5480	0.5531	0.5298
SNN	125.6802	0.8096	0.78	0.81	0.78

Starting with the analysis of the CNN results, the training time is decreased when transitioning from a single thread to 32 threads having an increase in performance of 28.45%.

when a single GPU is utilized the training time is also reduced having an increase of 38.44% in comparison to one thread configuration and 13.95% in comparison to 32 thread configuration. The test results and the improvement using a single GPU in comparison with all the capabilities of the CPU cores are there but is not a big difference, this means that the model again is not complex enough to make use of the hardware capabilities and that is the reason why the model it is not scaling with bigger improvement, this is also proven with the use of 2 GPUs because the training time goes up slightly to 95.3s.

If it is compared the results using CNN and a single Nvidia Tesla M60 GPU with the results obtained in Section 6.1.2, the difference in performance in the training time between both GPUs is 16.57% in favor of the Nvidia RTX 2060 which can be due to the newer architecture, higher frequency, and faster VRAM.

An abnormal behavior happened whit the SNN model when using the configuration of CPU with 32 threads showing less accuracy, recall, precision, and F1 score.

6.3 Discussion

To analyze with more detail, the results obtained during the tests in the different scenarios next it will be compared the training times for the detection and classification of network threats only for CNN and SNN ML models. The reason to show and compare only the results of these two ML models is that are the only ones that proved to be scalable using multithreading and/or GPUs.

Table 15 shows the time to train when CNN and SNN are implemented for the detection of network threats (binary classification) and Figure 9 will show graphically how it behaves.

Table 15: Time to train (binary classification)

Hardware configuration	CNN training time	SNN training time
Nvidia Tesla M60 1 GPU configuration	97.6922	101.6916
Nvidia RTX 2060 mobile 1 GPU configuration	97.9310	87.2675
Nvidia Tesla M60 2 GPU configuration	123.2225	131.0704
EC2 instance CPU 32 thread configuration	142.0379	35.1155
Ryzen 4600h 12 thread configuration	191.8092	21.4814
EC2 instance CPU 1 thread configuration	209.0336	35.7686
Ryzen 4600h 1 thread configuration	233.2915	27.7485

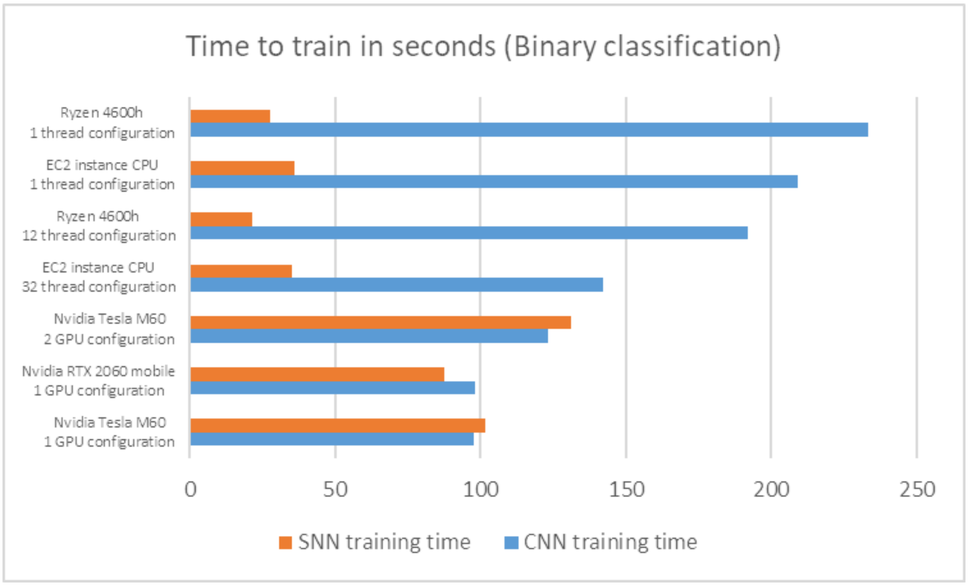


Figure 9: Time to train binary classification SNN and CNN.

Performance of the CNN in training times using single GPU configurations shows that the Nvidia Tesla M60 and Nvidia RTX 2060 mobile have similar training times (97.69s and 97.93s respectively). The minor differences could be attributed to the GPU architecture and memory bandwidth, but when is used multiple GPU configurations the times increase. This could be due to the overhead of synchronizing data and gradients between the GPUs. When CPU configurations are used, the performance of the CNN in training times for single thread configuration, the Ryzen 4600h time to train is 233.29s. On the EC2 instance with just one thread, the training time stands at 209.03s showing that the CPUs of the EC2 instance have better performance core by core. For multithread configurations on the EC2 instance with 32 threads, the training time is 142.04s. While moving to a Ryzen 4600h with 12 threads the training time is 191.81s. This makes sense as the EC2 instance has more cores. The multithread test showcases the benefit of using multithreading approaches.

For the SNN model in single GPU configurations, the Nvidia RTX 2060 mobile outperforms the Nvidia Tesla M60 in training time with 87.27s vs. 101.69s respectively. The RTX 2060 mobile could have architectural improvements that make it more efficient for this workload, and with multiple GPU configurations shows similar results to the CNN model, using two Nvidia Tesla M60 GPUs results in an increased training time for the SNN model (131.07s). For CPU configurations, the SNN model shows better results when multiple threads are involved. The EC2 instance with 32 threads achieves a training time of 35.12s, and the Ryzen 4600h with 12 threads performs even better with a training time of just 21.48s. The training times increase for single-thread configurations, but not much in comparison to the CNN case. On the EC2 instance with one thread, the training time is 35.77s, while on the Ryzen 4600h, it's 27.75s which are close results but this difference is something to test further in the future to validate the performance in different CPU architectures.

Table 16 shows the time to train for the classification of network threats (multi-class classification) and Figure 10 represents the results graphically.

Table 16: Time to train (multi-class classification)

Hardware configuration	CNN training time	SNN training time
Nvidia RTX 2060 mobile 1 GPU configuration	55.5871	75.3286
Nvidia Tesla M60 1 GPU configuration	66.6308	95.1097
EC2 instance CPU 32 thread configuration	77.4408	33.1640
Nvidia Tesla M60 2 GPU configuration	95.2998	125.6802
Ryzen 4600h 12 thread configuration	96.9567	21.5749
EC2 instance CPU 1 thread configuration	108.2400	35.3595
Ryzen 4600h 1 thread configuration	115.5832	26.6075

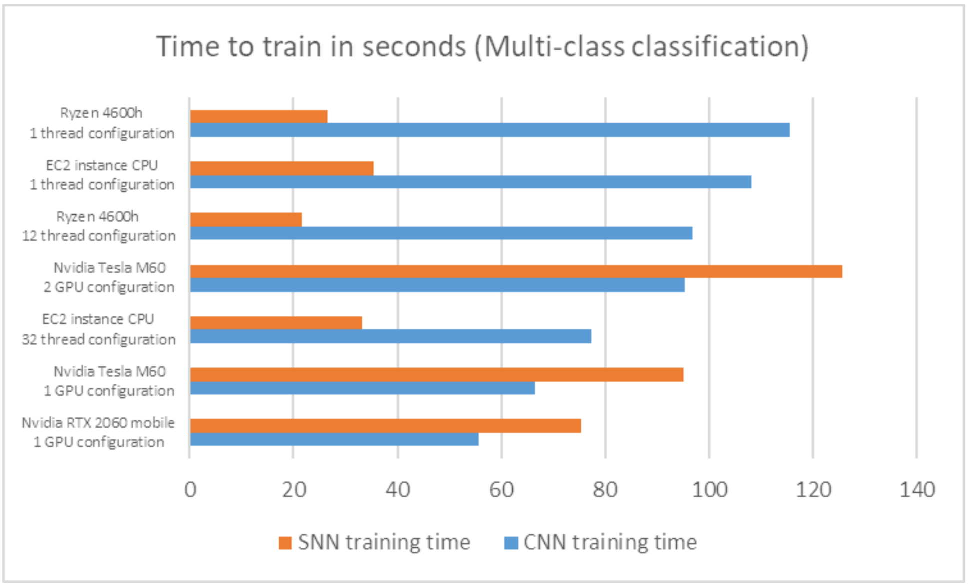


Figure 10: Time to train multi-class classification SNN and CNN.

Test of the CNN model in the classification of network threats shows that the Nvidia RTX 2060 mobile GPU consistently outperforms the training times of the Nvidia Tesla M60 and any other configuration using CPU. Scaling to two GPUs on the Tesla M60 doesn't linearly improve efficiency. On the CPU side, the 32-thread EC2 instance shows better results in comparison to a single-threaded configuration.

For the SNN model, the Nvidia RTX 2060 mobile still outperforms the Tesla M60 but shows to be slower in comparison to single and multi-threaded CPU configurations. The 12-threaded Ryzen 4600h manages to outdo most other setups being the multithread configuration the best performer for this ML model. Again, when moving to the dual Nvidia

Tesla M60 GPU configuration, the training time increases. The observations suggest that for certain tasks and models like the SNN, a well-chosen CPU configuration might deliver better or comparable performance to GPUs.

The approach followed was correct but needs more work to do to compare in a fair scenario each ML model. This is because even if the use of neural network algorithms where tested in the same hardware, the building of the code was done having the goal to reach the highest possible accuracy with the minimum amount of errors and not configuring them with comparable complexity in the model, leaving some questions unsolved.

Also, the number of possible errors in the classification of network threats can be difficult to deal with in a real scenario, because can produce a large number of false positives so the best approach in a real scenario is to use conditional statements after a network threat is identified because some models showed almost 100% accuracy and precision and then pass the workload to the classification stage. This approach can reduce a large number of false positives. This can be proved with the confusion matrix in Figure 11 for the classification of network threats (multi-class classification) where it shows error even in the “Normal” classification except for the DTC model where predicted all the “Normal” traffic is correct.

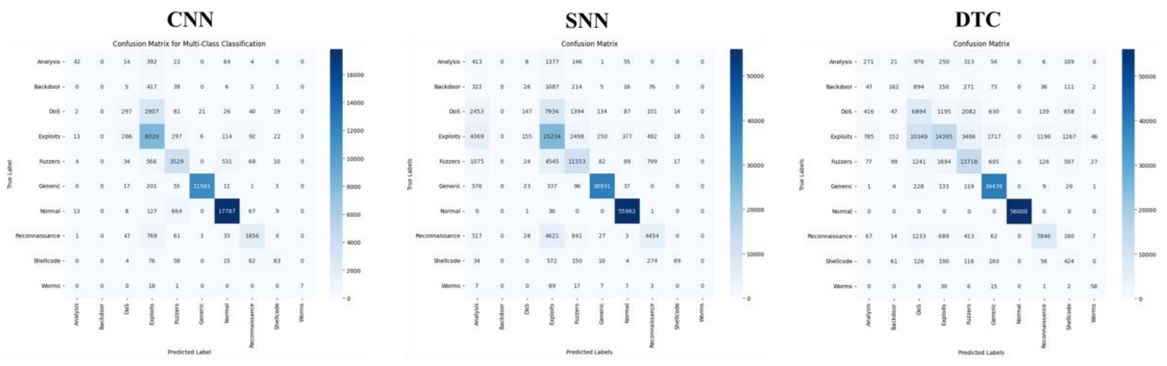


Figure 11: Confusion matrix CNN, SNN, and DTC.

The confusion matrix also proves that the CNN model will perform better than the other models on each classification having better precision on each class of network threats.

Finally, it is worth mentioning that even if the dataset is big enough to perform some tests and prove how the ML models scale it is important to work with a bigger dataset or create a more complex neural network model to push the limits of the hardware available and provide more information to measure the scalability of the models in larger scenarios.

7 Conclusion and Future Work

During the research of how machine learning algorithms for network traffic analysis can be configured not only to take advantage of parallelism, particularly for the detection and classification of threats, the role of hardware optimization carries more importance not only because of the hardware capabilities but also because the chip architecture can affect how well it performs in certain ML models. Both GPU parallel data processing and multithreading using a multicore CPU reduces training times providing information that can be helpful in accelerating the deployment of threat detection models. The parallel processing strengths of GPUs underscore their role in handling computationally intensive models such as CNNs and

SNNs. At the same time, multicore CPUs reveal that their architectures remain relevant in this domain, proving that they are optimized for the specific model demands.

The incorporation of scalable techniques into cybersecurity is crucial to develop new techniques to detect and classify attacks faster with a minimum amount of errors and using all the capabilities of the hardware available reducing the possible costs of energy, space, and maintenance required if the architecture is oversized. These techniques, designed for handling large volumes of data can accelerate the processing of it, but their possible effect with accuracy is intricate. For instance, while some ML models had low training times, their performance on multi-class tasks was occasionally compromised. Acceleration must not overshadow the criticality of model accuracy, especially in the domain of network traffic analysis, which contains sensitive information and the consequences of misclassification or detection are high.

Future work can be done by exploring hybrid configurations that harness both GPUs and CPUs in tandem which can lead to offering a hybrid ML model which provides both aspects, speed and accuracy. Lastly, as network threats evolve in complexity, ensuring that the ML models not only keep pace but anticipate future threats will be crucial. Continual learning and reinforcement learning methodologies, which allow models to learn and adapt over time, may offer promising results in this dynamic landscape.

In conclusion, the breach between hardware optimization and scalable machine learning techniques is one important aspect when designing new technologies for advancing threat detection in network traffic, and is because of it that the use of cloud services is a huge advantage because the models can be put into test in multiple hardware architectures, make them scalable and then eliminate everything not needed after the tests leaving only the architecture that suits best to the needs. The challenge is not just about speed but about ensuring a correct use of energy making the model efficient, accurate, and adaptable.

References

Tiwari, G., & Jain, R. (2022). Detecting and Classifying Incoming Traffic in a Secure Cloud Computing Environment Using Machine Learning and Deep Learning System. IEEE 7th International Conference on Smart Cloud (SmartCloud), 16-21.

Chkirbene, Z., Hamila, R., Erbad, A., Kiranyaz, S., Al-Emadi, N., & Hamdi, M. (2021). Cooperative Machine Learning Techniques for Cloud Intrusion Detection. 2021 International Wireless Communications and Mobile Computing (IWCMC), 837-842.

Salman, T., Bhamare, D., Erbad, A., Jain, R., & Samaka, M. (2017). Machine Learning for Anomaly Detection and Categorization in Multi-Cloud Environments. IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), 97-103.

Ilievski, A., Zdraveski, V., & Gusev, M. (2018). How CUDA Powers the Machine Learning Revolution. 2018 26th Telecommunications Forum (TELFOR), 420-425.

Divakar, S., Priyadarshini, R., & Mishra, B. K. (2020). A Robust Intrusion Detection System using Ensemble Machine Learning. 2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE), 344-347.

Salman, T., Bhamare, D., Erbad, A., Jain, R., & Samaka, M. (2017). Machine Learning for Anomaly Detection and Categorization in Multi-Cloud Environments. IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), 97-103.

Pwint, P. H., & Shwe, T. (2019). Network Traffic Anomaly Detection based on Apache Spark. 2019 International Conference on Advanced Information Technologies (ICAIT), 222-226.

Moustafa, N., & Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). 2015 Military Communications and Information Systems Conference (MilCIS), 1-6.

Eltaeib, T., & Islam, N. (2021). Taxonomy of Challenges in Cloud Security. 2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), 42-46.

Hung, C. -L., & Wang, H. -H. (2014). Parallel botnet detection system by using GPU. 2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS), 65-70.

Haidar, G. A., & Boustany, C. "High Perception Intrusion Detection System Using Neural Networks." 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, 2015.

Aljohani, A. & Bushnag, A. "An Intrusion Detection System Model in a Local Area Network using Different Machine Learning Classifiers." 2021 11th International Conference on Advanced Computer Information Technologies, 2021.

Chien, S. W. D., Markidis, S., Olshevsky, V., Bulatov, Y., Laure, E., & Vetter, J. (2018). Characterizing Deep-Learning I/O Workloads in TensorFlow. 2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS), 54-63.

Chien, S. W. D., Markidis, S., Olshevsky, V., Bulatov, Y., Laure, E., & Vetter, J. (2019). TensorFlow Doing HPC. 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 509-518.

Sattar, N. S., & Anfuzzaman, S. (2020). Data Parallel Large Sparse Deep Neural Network on GPU. 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 1-9.

Gupta, K. G., Maity, S. K., Das, A., & Wandhekar, S. (2021). Performance analysis of different distribution of Python and TensorFlow to efficiently utilize CPU on HPC Cluster. 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET), 1-6.

Ramirez-Gargallo, G., Garcia-Gasulla, M., & Mantovani, F. (2019). TensorFlow on State-of-the-Art HPC Clusters: A Machine Learning use Case. 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 526-533.

Quang-Hung, N., Doan, H., & Thoai, N. (2020). Performance Evaluation of Distributed Training in Tensorflow 2. 2020 International Conference on Advanced Computing and Applications (ACOMP), 155-159.

Campos, V., Sastre, F., Yagües, M., Torres, J., & Giró-I-Nieto, X. (2017). Scaling a Convolutional Neural Network for Classification of Adjective Noun Pairs with TensorFlow on GPU Clusters. 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 677-682.

Bagby, T., Rao, K., & Sim, K. C. (2018). Efficient Implementation of Recurrent Neural Network Transducer in Tensorflow. 2018 IEEE Spoken Language Technology Workshop (SLT), 506-512.

Malik, A., Lu, M., Wang, N., Lin, Y., & Yoo, S. (2018). Detailed Performance Analysis of Distributed Tensorflow on a GPU Cluster using Deep Learning Algorithms. In 2018 New York Scientific Data Summit (NYSDS) (pp. 1-8). IEEE. doi: 10.1109/NYSDS.2018.8538946.

Jain, A., Awan, A. A., Anthony, Q., Subramoni, H., & Panda, D. K. (2019). Performance Characterization of DNN Training using TensorFlow and PyTorch on Modern Clusters. In 2019 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 1-11). IEEE. doi: 10.1109/CLUSTER.2019.8891042.