A Lightweight AI-Based Intrusion Detection System Optimised For Resource-Constrained Networks

Christopher Blignaut
The University of Cape Town
Cape Town, South Africa
blgchr003@myuct.ac.za

ABSTRACT

The proliferation of Internet of Things (IoT) devices and network interconnectivity has created a need for effective security solutions in resource-constrained networks. AI-based Intrusion Detection Systems (IDSs) are able to scale and adapt towards the increasing volume of network traffic and sophistication of modern cyber attacks, however, their high computational demands make them impractical for low-resource environments. The LAIDS project addresses this issue by proposing three lightweight AI-based IDS (LAIDS) architectures optimised for resource-constrained environments. The study compares the performance of a CNN-GRU and a GAN-based hybrid deep learning model against a baseline 1D and 2D CNN-based model. The proposed architectures are evaluated on their multi classification ability, ability to detect zero-day attacks and their resource efficiency through post-training quantization. Our findings demonstrate that these lightweight AI models can maintain reasonable threat detection accuracy and lower false positive rates while maintaining the strict computational and memory footprint required for deployment in resource-constrained environments, thereby offering a scalable and effective security solution for resource-constrained networks.

1 INTRODUCTION

The increased usage of Internet of Things (IoT) devices and increase of network interconnectivity has created a need for effective security solutions. One such security solution is the Intrusion Detection System (IDS) [13, 25]. The purpose of an IDS is to serve as a warning system for when the network becomes the target of a cyber attack [13]. This is done by classifying network traffic as either normal or malicious and notifying network administrators if the traffic is classified as the latter [36]. Traditional IDSs lack the scalability and adaptability to handle both the ever-increasing volume of network traffic and the complexity of cyber attacks [7, 25] . As such, IDSs have been enhanced with AI. Supervised deep learning models like Convolutional Neural Networks (CNNs), Gated Recurrent Units (GRUs) and Multilayer Perceptrons (MLPs) have been shown to effectively classify network traffic [25, 36]. However, they require data to be processed manually whilst training. Deep learning models like Generative Adversarial Networks (GANs) are able to manually adapt and classify without manual intervention, however they are more resource intensive [2, 28]. While, these AI-based IDSs (AI-IDSs) are able to both adapt to changing network traffic and scale with the increasing traffic volume, they are resource intensive and thus, struggle when placed in resource-constrained network environments like IoT and edge networks [19, 20]. This has left a security gap exposing networks to cyber attacks through their resource-constrained attack surfaces [32]. Research into optimising AI-IDSs for lightweight deployment is paramount in ensuring that these gaps are protected.

2 RESEARCH OBJECTIVES

The vulnerability of low-resource environments such as IoT networks, edge devices, and embedded systems has become a critical area of research. Existing IDS solutions either lack the ability to detect new attacks, otherwise known as zero-day attacks, or are too computationally demanding for lightweight deployment leaving low-resource networks to be particularly vulnerable [7, 19]. As such, there is a need for research into a lightweight, AI-driven IDS capable of zero-day attack detection in real time, while operating within resource-constrained environments.

In order to address this problem, the LAIDS project aims to develop a Lightweight AI-IDS (LAIDS) for deployment on network gateways or personal computers to classify network traffic and identify cyber attacks in resource-constrained environments. To do this, we developed a series of IDSs and evaluate their performances comparatively. This paper's aim is to propose 2 candidate IDSs namely, a GAN-based IDS with an auxiliary MLP multi classifier and a CNN-GRU-based IDS. These proposed models are to be evaluated against both a benchmark 1D and 2D CNN-based IDS to determine if these proposed models are suitable for real-world deployment. As such, this paper sets out to answer the following research question:

How can a Hybrid CNN-GRU or GAN-based IDS outperform a CNN-based IDS in terms of zero-day attack detection, detection accuracy, false positive rate, and computational resource usage in a resource-constrained network environment?

3 BACKGROUND & RELATED WORK

In this section, we will provide the reader with an overview of the models used. This will be followed by a summary of the current academic literature when developing a lightweight IDS.

3.1 Background

3.1.1 *Convolutional Neural Network.*

Convolutional Neural Networks (CNNs) are supervised deep learning models. They have demonstrated their ability to consistently, accurately classify network traffic [24, 26, 35]. As such, they have seen extensive application as IDSs due to their ability to recognise complex spatial patterns [26]. A drawback of employing a CNN is that the operations required to perform multi classification are resource intensive and therefore, CNNs struggle in resource-constrained environments [34]. That being said, the combination of

a CNN with another model architecture eliminates this drawback [17].

3.1.2 *Gated Recurrent Unit.*

Gated Recurrent Units (GRUs) are a type of Recurrent Neural Network which specialise in capturing temporal features [31, 41]. GRUs offer a more computationally efficient alternative to Long Short Term Memory (LSTM) units through the simplification of the gating mechanism resulting in a more streamlined design [5, 41].

3.1.3 *Multilayer Perceptron.*

Multilayer Perceptrons (MLPs) are forward propagation neural networks which consist of an input layer, output layer and a series of intermediate hidden layers [4]. They capture complex attack patterns though non-linear modelling and back propagation-based learning [4]. Additionally, they are the smallest and most lightweight model of all the models mentioned in this paper, making them the most suitable model for deployment in resource-constrained environments [38].

3.1.4 *Generative Adversarial Network.*

Generative Adversarial Networks (GANs) are made up of two components: a generator which creates synthetic network patterns and a discriminator which evaluates and classifies incoming network traffic as either genuine or generated [28]. In the context of an IDS implementation, the discriminator can be deployed to evaluate incoming network traffic and differentiate normal from abnormal behaviours [26]. This approach does not rely on known attack signatures making GANs particularly effective for detecting zero-day attacks [26]. GANs can be optimised for low-resource environments with a variety of techniques such as quantization and only deploying the discriminator component to the network [18].

3.1.5 *Quantization.*

Post-training quantization is the process of converting a trained, full-precision neural network into a lower bit-width representation in order to reduce the computational and memory requirements of the model [40]. Post-training quantization can be achieved through two approaches: dynamic quantization, in which only the model weights are quantized while activations remain at full precision, and full quantisation, in which both the model weights and activations are quantized [23]. While full quantization can reduce the resource requirements of the model, it also makes the model more sensitive. If the model activations are not properly retrained, it may suffer from class collapse resulting in a degradation of performance [14]. In the context of resource-constrained environments, quantization is valuable as it enables IDS models to maintain strong performance while significantly reducing resource overhead [23, 40].

3.2 Related Work

Recent studies have demonstrated that AI-IDS outperform traditional IDS in terms of zero-day attack detection and detection accuracy [25, 36]. Given the significance of an IDS in regards to the security of a network, extensive research has been conducted into the application of AI-IDSs [25]. However, the comparison of the results from separate works proves to be difficult as they employ different datasets when measuring the performance of their models. Fortunately, many papers benchmark their model's performance using the CICIDS 2017 dataset which was developed by Sharafaldin et al. [29]. This dataset consists of labelled network flows rather than individual packets. This is preferable as network flows provide a richer set of aggregated features, enabling the models to perform a more comprehensive analysis of the network traffic. In order to provide a fair comparison, this study reports only on prior work where IDSs were evaluated on network flows rather than individual packets.

Reddy et al. [26] noted that CNNs excelled at processing high dimensional data and recognising complex patterns, a skill required for detecting malicious traffic. Udurume et al. [35] compared the viability of a range of AI models towards IDS implementation. They found that their CNN model achieved a binary classification accuracy of 92.6% and a multi-class classification accuracy of 94.8%. Likewise, Arsalan et al. [3] explored the practicality of a 1D CNN IDS implementation in which their proposed model achieved a 99.9% multi classification accuracy. Although 2D CNNs demand greater computational resources, their demonstrate potential in accurately classifying network traffic in resource-constrained environments. Pham et al. [24] proposed a lightweight 2D CNN IDS which analysed network traffic by transforming the network data into image representations. Through this approach, they were able to minimise the computational strain on the network demonstrating that there is potential for a 2D CNN IDS implementation in resource-constrained environments.

Choosing to leverage the spatial feature extraction ability of a CNN with the temporal feature extraction ability of a GRU, Cao et al. [5] proposed a CNN-GRU based IDS. The model was evaluated on the CICIDS 2017 dataset. To combat the imbalance of the dataset, they employed ADASYN oversampling as well as random forest feature selection. The proposed model achieved a multi classification accuracy of 99.65% demonstrating the applicability of a CNN-GRU based IDS.

While there are fewer studies into the implementation of a GAN-based IDS than the models proposed above, GANs still show promise in an IDS implementation. Shahriar et al. [28] compared the performance of a GAN-based IDS to a stand alone IDS. Both models were trained and evaluated on the KDDCup99 dataset. From this they found that the GAN-based IDS yielded a higher F1 score than the stand alone artificial neural network-based IDS, noting that the GAN-based IDS did not suffer from the class imbalance present in the dataset. Park et al. [22] explored the use of GANs in IDS implementations. They found that GANs can learn new attack signatures through synthetic data generation which in turn improves the robustness of the model against novel attack vectors. Ali et al. [2] further analysed the incorporation of a GAN in an IDS implementation, particularly in a resource-constrained environment for IoT application. Their findings demonstrated that the use of adversarial training resulted in a highly accurate IDS which was more resource efficient than typical GAN-based IDS implementations. However, they did note that more research must be conducted into the lightweight optimisation of a GAN-based IDS.

Studies have shown the potential of the implementation of a MLP-based multi classifier in an IDS context. Cherfi et al. [9] proposed an MLP model which incorporated the techniques: simulated annealing and adaptive large neighbourhood search in order to increase the model's efficiency. When evaluated against the CICIDS 2017 dataset, the model achieved an accuracy of 99.80% demonstrating the potential of a MLP-based lightweight IDS implementation. Similarly, Rosay et al. [27] implemented an MLP based IDS to answer for the shortcomings of machine learning based IDS implementations. When benchmarked against the CICIDS 2017 dataset, their proposed model achieved an accuracy of 99% and a false positive rate of 0.7%. That being said, the authors acknowledged that the model's resource requirement's were too great for viable real-world implementation.

4 EXPERIMENT DESIGN & IMPLEMENTATION

In this section we will present the experimental design and implementation of our study. We describe the methods we used in the dataset preprocessing, the model architectures and the hardware and software environments used to ensure experiment reproducibility.

4.1 Dataset

4.1.1 The CICIDS 2017 Dataset.

This study utilised the CICIDS 2017 dataset [29], which is widely used to train and evaluate AI models in cybersecurity research. The CICIDS 2017 dataset was created by The Canadian Institute for Cybersecurity [30]. It was created using real-time network traffic data which was generated in a lab setting and captured over the course of 5 days [36]. The dataset contains over one million pieces of network data in which the following attack vectors are found: Brute Force, Botnet, Denial of Service (DoS), Distributed Denial of Service (DDoS), Port Scan, and Web Attacks [39]. One benefit from the network traffic being lab generated is that no personal information is present in the data. This ensures that the data is both ethically sourced and does not need to be anonymised. This has resulted in a large feature set totalling 79 features [30].

Despite its advantages, the CICIDS 2017 dataset exhibits several issues including inconsistencies and redundant records. To mitigate these issues, we employed Ribeiro's preprocessed version of the CICIDS 2017 dataset[1]. Ribeiro cleaned and preprocessed the dataset, by taking the following actions: Ribeiro first merged the original dataset into a single CSV file. Ribeiro then removed any duplicate rows and columns as well as replacing any infinite values with 'NaN'. Ribeiro also removed any rows with missing values and removed leading and trailing white spaces for consistency. Additionally, Ribeiro employed a Kruskal-Wallis test and a Random Forest algorithm to remove statistically irrelevant features. Finally Ribeiro grouped similar attack labels together and removed the rare attacks Infiltration and Heartbleed to prevent potential over fitting and improve model generalisation.

[1]The dataset can be found here https://www.kaggle.com/datasets/ericanacletoribeiro/cicids2017-cleaned-and-preprocessed

4.1.2 Data Preprocessing.

When preprocessing Ribiero's cleaned dataset, we first split the data into training, testing and validation splits using stratified sampling to preserve class distribution. A Standard Scaler from Scikit-learn library was then fitted on the training split and subsequently used to transform all splits. Despite Ribeiro's efforts, the CICIDS 2017 dataset suffers from class imbalance, with benign network traffic making up 83.1% of the dataset. In order to address this class imbalance, we downsampled the benign traffic in the training split to 250,000 samples.

4.1.3 Adaptive Synthetic Sampling.

To further address the class imbalance, we employed Adaptive Synthetic (ADASYN) sampling. ADASYN is an oversampling technique which is used to create synthetic samples of minority classes to address class imbalance [12]. Unlike other oversampling techniques, ADASYN assigns greater weights to the majority class and thus focuses more on generating synthetic samples which are similar to the majority class[12]. This can be seen in Figure 1 which shows how the oversampled bright red samples are generated from samples which were similar to the blue majority class. By generating samples which bear a closer resemblance to benign traffic, the model's ability to distinguish subtle differences between normal and malicious network traffic is enhanced, thereby improving attack detection.

ADASYN oversampling was applied to the training splits of both the 1D CNN, 2D CNN and the MLP model ensuing that all minority classes were scaled up to 200 000 samples. This choice of partial balancing, rather than fully equalising all classes, was made to reduce the risk of over fitting on synthetic data while still improving minority class representation. No ADASYN oversampling was employed when creating the training split for the GAN portion of the GAN-MLP model, as the GAN was only trained on benign traffic. The same holds true when creating the training split for the CNN-GRU as this resulted in the best overall performance of the model. Additionally, no ADASYN oversampling was employed on the validation and testing splits of all models to ensure that the models were only evaluated on real-world traffic distributions.

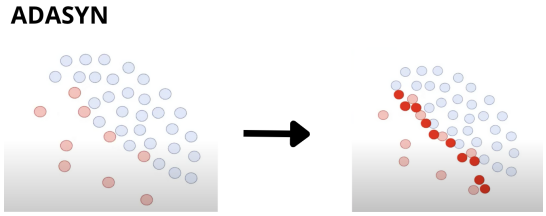


Figure 1: ADASYN Oversampling.

To ensure the reproducibility of our results, we saved the artifacts of our data preprocessing, these being: the split indices, scaler, label encoder, feature names, ADASYN oversampled training split, testing and validation data split as well as the random number generator seeds. These artifacts were loaded in each time when preprocessing the data for our models.

4.2 Models

4.2.1 *Baseline 1 Dimensional CNN.*

CNNs routinely demonstrate effective performance when classifying network traffic [26, 36]. Lightweight implementations are to accurately classify network traffic and detect threats [33]. As such, we chose to benchmark our proposed models against a 1D CNN based IDS.

As can be seen in Figure 2, this model's architecture consists of two one-dimensional convolutional layers with the first layer consisting of 32 filters and a kernel size of 2. The second layer consists of 16 filters and a kernel size of 3. Both of these layers employ a ReLu activation function and are followed by batch normalisation, which served to provide stability during the training process. This is then followed by max pooling layers which serve to reduce the feature maps, making the model more lightweight. This feature extraction process is flattened and then passed through a fully connected dense layer comprising of 64 neurons and a ReLu activation function. While it does go against convention to have a dense layer with a larger number of neurons after a convolutional layer with a smaller number of neurons, we found that this hyper parameter setup was optimal through Bayesian optimisation [11]. To reduce over fitting, L2 regularisation is applied to the dense layer which is followed by a dropout 0.5. Finally, the model employs a soft max output layer to produce a range of normalised class probability distributions, enabling multi class classification. The model is compiled using an Adam optimiser and a sparse categorical cross-entropy loss function.

In order for us to train the CNN, we first loaded in saved artifacts as mentioned in Section 4.1. We then reshaped the data by adding an extra dimension resulting in our data having the shape of (samples, features, timesteps). This allowed the data to be a 2 dimensional object, the input shape required for a 1D CNN.

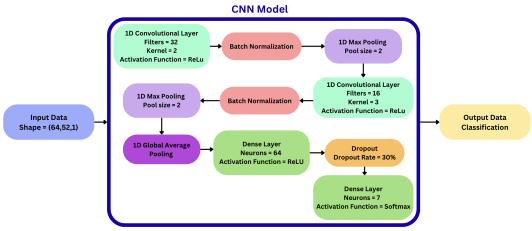


Figure 2: Baseline CNN Model Architecture

4.2.2 *2 Dimensional CNN.*

In addition to the baseline 1D CNN, we also sought to determine the effect which analysing the data in a second dimension would have on the model's performance. As such, we created a 2D implementation of our baseline 1D CNN model as shown in Figure 3. This model closely resembles the 1D baseline with a few key changes in the model's architecture. The first change being that the 1D convolutional layers have been changed to 2D convolutional layers. The batch normalisation and 1D max pooling layers found after the first convolutional layer have been removed as the data has been reduced too much from the first 2D convolutional layer, resulting in negative dimensions when max pooling is applied. The remaining 1D max pooling and global average pooling layers have been converted to 2D max pooling and 2D global average pooling layers to account for the 2 dimensional data. The 2D CNN also requires a 2 dimensional input shape. As such we reshaped the data into the shape of (height, width,colour), an image with the height of 7, the width of 8 and a channel of 1 corresponding to greyscale.

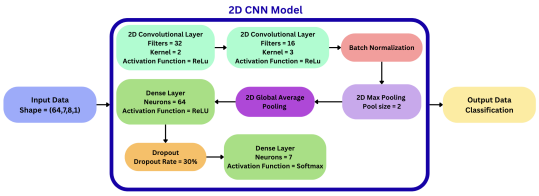


Figure 3: 2D CNN Model Architecture

4.2.3 *CNN-GRU.*

As established above, CNNs excel at extracting spatial data and using it to classify network traffic [26, 36]. Combining this ability with the temporal feature extraction capabilities of a GRU will allow for an IDS which will be able to detect more complicated, time based attacks [5].

This model consists of both a 2D CNN and 48 GRU units as seen in Figure 4. The 2D CNN is applied in a time-distributed manner to the individual frames of the sequenced samples in order to extract the spatial level features of each frame. The extracted features of the time series are then fed into the GRU units for temporal modelling. The CNN employs a 3 block structure as shown in 5. The first convolutional block contains 24 filters with a kernel size of 7, and L2 regularization. Batch normalisation and average pooling are applied to reduce data dimensionality, optimising the model for lightweight deployment. A dropout of 0.4 is then applied to prevent over fitting. The second convolutional block follows the same structure as the first convolutional layer containing 48 filters with a kernel size of 5 in order to detect fine grain features. The third block applies a convolutional layer consisting of 72 filters, a kernel size of 3 and L2 regularization. This is followed by batch normalisation and a dropout of 0.4. A global average pooling layer is then applied to aggregate spatial feature maps into compact frame-level vectors.

The GRU layer, consisting of 48 units, processes these sequenced vectors, capturing temporal features. The output is then passed into a dense layer with a soft max activation function to classify the network traffic. The model is compiled using an Adam optimiser and a sparse categorical cross-entropy loss function.

Once the CNN-GRU dataset was loaded in, the data was combined into sequences of 10, so that the GRU unit would be able to leverage its temporal feature extraction. After which, an extra dimension was added before being reshaped for the 2D CNN resulting in the

data having an input shape of (samples, features, timesteps, height, width, colour).

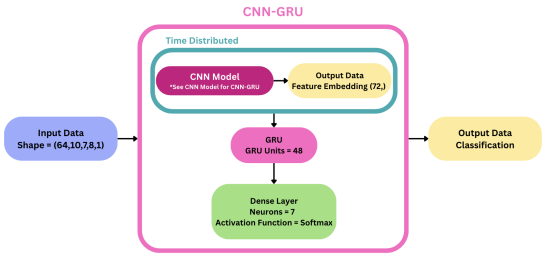


Figure 4: CNN-GRU Model Architecture

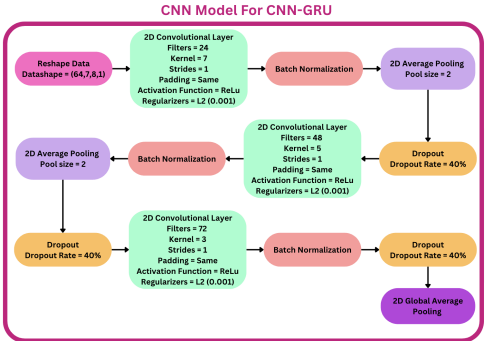


Figure 5: CNN Model for CNN-GRU

4.2.4 GAN.
The discriminator portion of the GAN serves as a binary classifier for the IDS. If the network traffic is flagged as potentially malicious by the discriminator, then an auxiliary MLP multi classifier is employed to further classify the network traffic as seen in Figure 6. This approach benefits from a lack of reliance on known attack signatures [22, 26]. This in turn should make the model particularly effective for detecting zero-day attacks, a feature paramount to the quality of the IDS.

The discriminator network is comprised of 2 dense layers, each consisting of 128 neurons with a LeakyReLU activation function where $\alpha = 0.2$ and is followed by a dropout rate of 0.3 as can be seen in Figure 6. This is followed by a final dense layer consisting of 1 neuron with a sigmoid activation outputting the discriminator's probability of the input data being real or synthetic. Binary cross-entropy is used to calculate the adversarial loss. The discriminator was trained using a GAN. To do this, we constructed a generator which is comprised of 3 dense layers: the first with 128 neurons and the second with 256 neurons. Both of these layers are followed by a LeakyReLU activation (with $\alpha = 0.2$) and batch normalisation. The final layer consists of 52 neurons and employs a hyperbolic tangent (tanh) activation to ensure its output is scaled further mimicking the preprocessed real network traffic. The generator's loss function is defined as the cross-entropy loss obtained when generated samples were classified as real. Both the generator and discriminator networks are trained using the Adam optimiser ensuring stable convergence during adversarial training.

The MLP multi classifier consists of 3 blocks. The first block consists of an input dense layer with 256 neurons and the second block consists of an input dense layer 128 neurons. Both blocks contain a ReLU activation function and L2 regularisation to mitigate over fitting. This is followed by a dropout layer with a rate of 0.2 to encourage generalisation. The final layer consists of 7 neurons and employs a soft max function to produce a range of normalised class probability distributions, enabling multi class classification. The model is compiled using an Adam optimiser and a sparse categorical cross-entropy loss function.

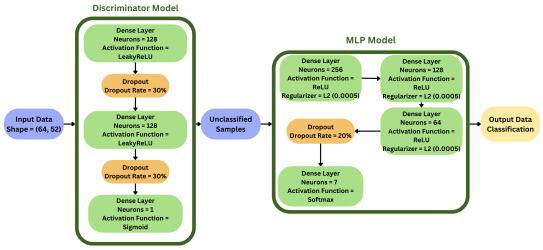


Figure 6: GAN-MLP Model Architecture

4.3 Evaluation Metrics

To determine if real-world deployment of the proposed LAIDSs is viable we evaluate our models using 3 criteria: multi classification performance, resource usage and zero-day attack detection. Below is an explanation of the evaluation metrics used

4.3.1 Multi classification Performance Metrics.
When evaluating our models multi classification performance, we report the model's accuracy, precision, F1 score, recall and False Positive Rate (FPR). The combined use of these metrics provides a comprehensive view on the performance of our models. To ensure our findings are accurate, the data presented is the weighted average of the score metric. This takes into account any class imbalance from the test dataset. It is calculated as follows:

$$\text{Metric}_{\text{weighted}} = \frac{\sum_i \text{Metric}_i \cdot \text{No.Samples}_i}{\sum_i \text{No.Samples}_i} \quad (1)$$

4.3.2 Resource Usage Metrics.
When evaluating the resource requirements of our models we employ 2 metrics. The first metrics measure is the Resident Set Size (RSS) which is the amount of space in RAM that the model occupies [8]. This offers a reliable estimate of the model's runtime memory requirements. We also measure CPU utilisation as it provides insight into the processing load imposed onto the CPU by the model during inference. Excessive CPU usage may degrade overall system performance which is why we chose to employ this metric. These metrics illustrate the resource consumption of the models which can determine whether lightweight deployment of our proposed models is feasible.

4.4 Hardware & Software Used

The models for this project were written in Python using the Tensorflow framework. They were developed in both Google Colab and Jupyter notebooks which were run on our native machines. The Scikit-learn, Joblib and Pandas libraries were used to preprocess the data. The NumPy library allowed for calculations made by the models. The Tqdm and Tabulate libraries were used for visual add-ons. The confusion matrices presented in the appendix section were created using the seaborn library. The graphs presented in the results section were created using Microsoft Excel. These models were developed and trained on both Google Colab and an M1 macbook which allowed the project to leverage more computing resources. Google Colab provided access to GPUs, specifically the NVIDIA T4 GPU and the Intel Xeon CPU which was useful for training and testing. The local development environment on an M1 macbook was used for initial coding and testing.

4.5 Ethical and Legal Issues

This study did not experience any legal concerns or require ethical clearance as all experiments were conducted on the CICIDS 2017 dataset, which was generated in a controlled lab environment and thus contains no personal information. The dataset has been widely adopted in academic research and is publicly available [30]. All code, models, and supporting documentation produced as part of this project have been released as open-source artifacts to promote transparency and reproducibility.

5 EXPERIMENT METHODOLOGY

In order to evaluate the feasibility of real-world lightweight deployment of our proposed LAIDS, we designed 3 experiments. Experiment 1: Multi Classification Performance which evaluates the baseline models' ability to classify network traffic and detect threats. Experiment 2: Post-Training Quantisation evaluates the resource consumption of our baseline models and examines the impact that post-training quantization has on model resource consumption and performance. Experiment 3: Zero-Day Attack Detection evaluates our models' capacity to generalise and detect zero-day attacks. The models used in Experiments 2 and 3 stem from modifications made to the pre-existing baseline architectures. Collectively, these experiments provide a comprehensive evaluation of our proposed models from which we are able to determine the practicality of lightweight deployment.

5.1 Experiment 1: Multi Classification Performance

The purpose of this experiment is to determine the multi classification ability of our proposed models and to establish a baseline performance for each of the proposed architectures. By providing a benchmark, we provide a fair comparison of each of the proposed architectures as well as aid in demonstrating the effects of quantization on each of the models.

All of the models were trained on the datasets as described in Section 4.1. For each individual model, the data was loaded and then reshaped into the model's required input shape. After preprocessing the data, all the models, apart from the GAN portion of the GAN-MLP were trained, employing callbacks which monitored validation accuracy and early stopping which monitored validation loss. These measures were used to prevent over fitting. Due to the sensitive nature of training GANs, the best discriminator model produced during training was manually selected through the comparison of evaluation graphs produced during training. Once the models were trained, they were evaluated using the testing data split.

In order to provide a comprehensive analysis of the proposed baseline models' multi classification ability, we evaluated the models under the following metrics: accuracy, precision, recall F1 score and FPR.

5.2 Experiment 2: Post-Training Quantisation

The purpose of this experiment is to determine the effect that post-training quantization will have on model performance and resource consumption. Quantization is a key operation for model deployment when deploying in resource-constrained environments [37]. As quantization reduces the model's size, data is lost. As such, we aim to evaluate the effectiveness of the models post quantization.

The models which we quantized in this experiment are the baseline models, i.e. the baseline CNN, 2D CNN, CNN-GRU and GAN-MLP models. The models first underwent dynamic quantization where only the model weights were reduced in bit-width. The quantized models were then evaluated in multi classification performance using the same method as described in Experiment 1 to determine if the was any degradation of performance due to quantization. The baseline models then underwent full quantization in which the activation weights also have their precision reduced and were retrained on a small sample from the training set. This sample contains 8192 network flows and is initialised using a seed for reproducibility. These fully quantized models were then tested using the method as their dynamically quantized counterparts.

Both the baseline and quantized models were also evaluated on their computational resource requirements. To do this, we tested each model individually. First, we loaded the model and the dataset as described in Section 4.1. From this we extracted a single input sample from the test split. This sample was reshaped into the shape expected by the model architecture being tested. After reshaping, we had the model initially classify the test samples 5 times to warm up the cache before having the model further classify the same samples 10 more times, this time measuring the CPU usage and RSS of the model. After the measurements were taken, we proceeded to calculate the average CPU utilisation from the measurements taken by a background thread, which is the result found under the *CPU Utilisation (%)* metric. To calculate the RSS of the model, we took two measurements of the RAM: one before the model was loaded in for validation, and one after the model had been loaded in and its variables cleared. We calculated the RSS to be the difference between the measurement after the model was loaded into RAM and the measurement before the model was loaded into RAM. The result is found under the *Memory Overhead (MB)* metric.

We ensured that the runtime was refreshed and RAM cleared before evaluating each model.

5.3 Experiment 3: Zero-Day Attack Detection

The purpose of this experiment is to determine how well the model generalises towards unseen network attacks. As established, cyber attacks are continually evolving and becoming more complex [1, 25]. As such, any model deployment should take zero-day attacks into account.

In order to determine our models' ability to detect zero-day attacks, we took the following steps: First we retrained our baseline models, this time taking care to remove a single attack type from the training and validation sets while ensuring that it is found in the testing set. This removed attack type served as our zero-day attack. We then trained the models on this augmented dataset using the same training pipeline as described in the Multi Classification Performance experiment. After this, we evaluate the model's performance by determining how much of the zero-day attack the model detected.

We then quantized the retrained models following the same pipeline as described in the Post-Training Quantisation experiment, with the exception that we did not measure any of the model's resource usage.

For this experiment, we chose for the DoS and the DDoS attacks to serve as our zero-day attacks. The models were first trained with the DoS attack removed from the training set, their performance was evaluated and the results were recorded. The models and datasets were then reset and retrained with DDoS attack removed from the training set, their performance was evaluated and the results recorded.

6 RESULTS

6.1 Multi Classification Performance

Table 1: Performance Comparison of Models

Model Weights	Accuracy (%)	Precision (%)	F1 Score	Recall (%)	False Positive Rate (%)
1D CNN	95	98	95	96	6.1
2D CNN	96	98	97	96	3.8
CNN-GRU	99	99	99	99	0.003
GAN-MLP	93	96	94	93	5.2

All baseline models achieve a high degree of accuracy, precision, and a high F1 score, while achieving a low to moderate recall and FPR as seen in Table 1. The CNN-GRU model demonstrates the best overall performance as it achieves the best score in all metrics. This can be attributed to the combination of both the CNN's ability to extract spatial features from the network traffic combined with the GRU's ability to extract temporal patterns from the time distributed CNN. The combination of the two units allows for each one to compliment each other by providing strengths to each other's weaknesses. This has resulted in a model which has achieved a 99% in accuracy, 99% in precision, 99% in recall, an F1 score of 99% and 0.003% FPR. This demonstrates an IDS which is able to effectively detect threats and accurately classify them, while achieving a low number of false alarms. In terms of multi classification performance, this model is viable for real-world deployment.

The 1D CNN also demonstrated an effective multi classification performance by achieving an accuracy of 95%, a precision of 98%, a recall of 96%, and an F1 score of 95%. While this model demonstrates strong precision, indicating that it is highly effective in distinguishing benign from malicious traffic, its higher false positive rate of 6.1% indicates that this model flags a large amount of normal network traffic as malicious. This decrease in performance illustrates the drawback of relying solely on one dimensional spatial feature extraction as the model is unable to establish cross spatial patterns between features in its input matrix. Despite this, the 1D CNN remains a competitive lightweight IDS which demonstrates a sufficient multi classification performance making real world deployment feasible.

While deployment of the 1D CNN is feasible, the 2D CNN is the superior choice as it outperforms the 1D CNN in both accuracy and F1 score scoring 96% and 97% respectively. It does this while also achieving a lower false positive rate of 3.8%. This improvement can be attributed to the 2 dimensional convolutional kernels, which allow the model to better capture localised correlations in the spatial features and hierarchical patterns found in the network traffic. The introduction of an extra dimension has resulted in a more balanced model which provides an improved multi classification accuracy and fewer false alarms, making it particularly well-suited for deployment in environments where a low false positive rate is required.

The GAN-MLP achieved the worst performance among the models. This can be seen from the model's performance where it achieved an accuracy of 93%, precision of 96%, recall of 93%, an F1 score of 94% and a false positive rate of 5.2%. The reliance on the auxiliary MLP classifier was primarily responsible for the model's ability to identify anomalous traffic, whereas the discriminator component often hindered overall performance through misclassifying attacks as normal traffic. This is illustrated through the difference in performance as seen in the confusion matrices of the Discriminator and MLP found in Appendix A. The model's relatively low recall indicates that it is prone to misclassifying malicious traffic as benign allowing intrusions to take place. As such, in terms of multi classification performance, the GAN-MLP is not feasible for real-world deployment.

From this we can conclude that the CNN-baseline models all achieve a high degree of precision, recall, a low FPR and a high F1 score when classifying network traffic. As such they are feasible for real-world deployment. That being said, the performance of the GAN-MLP demonstrates that real-world deployment is not feasible as it allows too many attacks to slip through.

6.2 Post-Training Quantisation Experiment Results

All quantized versions of the baseline models uphold the same level of multi classification performance as their baseline counterparts, which is presented in Appendix B. The only exceptions being: the dynamically quantized Int8 GAN-MLP, which saw a minor increase in performance and the fully quantized Int8 models which experienced a reduction in multi classification performance. This can be attributed to the models undergoing class collapse due to improper retraining of the activations resulting in the model being more likely to classify network traffic as benign. Nevertheless, below are the results demonstrating the impact which post-training quantization has on the resource requirements of the models.

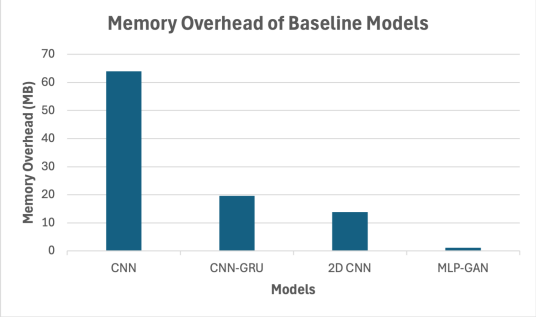


Figure 7: Memory Usage Comparison of Baseline Models

As seen in Figure 7, the baseline models demonstrate varying degrees of performance when measured on RSS. The baseline 1D CNN requires the most memory of the 3 models to perform its calculations at 64 MB. This is due to the model's expensive computations which are required to extract the individual spatial features from the data. This can be seen through the CNN-GRU's lower RSS of 19.6 MB despite being a more complex model as well as the 2D CNN's lower RSS of 13.8 MB. Leveraging inner-feature correlation, the 2D CNN is able to minimise its RSS as it does not have to process each feature individually. The CNN-GRU achieved a lower RSS as it was trained with a larger dropout compared to the baseline 1D CNN. The GAN-MLP achieved the lowest RSS of only 1.1 MB. This is due to the IDS's light weight architecture.

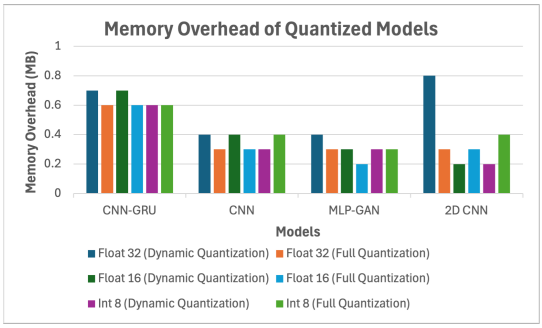


Figure 8: CPU Utilisation Comparison

Figure 8 demonstrates that all quantized models achieve a lower RSS compared to their baseline counterparts. This sizeable difference can be attributed to the pre-allocated tensor arenas which the quantized models use as opposed to the baseline models. These arenas forgo loading the full Tensorflow graph which significantly decreases the model's memory overhead [10].

The quantized models generally exhibit similar memory footprints, with reductions in RSS varying by architecture. The 1D CNN experienced the most dramatic decrease, with its memory usage falling from 64 MB to just 0.4 MB. The 2D CNN also showed a substantial reduction in RSS, reducing from 13.8 MB to 0.8 MB. Similarly the CNN-GRU dropped from 19.6 MB to 0.7 MB, while the GAN-MLP shrank from 1.1 MB to 0.4 MB. Among the quantized variants, the CNN-GRU consistently recorded the largest RSS, with a minimum of 0.6 MB. This value is larger than the quantized versions of the 1D CNN, 2D CNN, and GAN-MLP. This can be attributed to the complexity of the model's architecture.

The 2D CNN also presented a unique case as while it reduced well overall, the dynamic Float32 model recorded the highest quantized memory overhead at 0.8 MB. This is due to the 2D CNN's relatively high memory footprint which demonstrates the need for activation weight quantization to further minimise its memory usage. That being said, the Float16 and Int8 dynamic quantization models were compressed to just 0.2 MB, effectively reducing itself to the same size as the GAN-MLP, the smallest model overall.

All the fully quantized models apart from the 2D CNN variants achieved a lower RSS compared to their dynamically quantized counterparts. This reduction is due to the decrease of the model activations, thus lowering the memory requirements of the model. However, this trend did not hold for the fully quantized Int8 models, which in some cases matched or even exceeded the overhead of their dynamic counterparts. This can be attributed to interpreter overhead in TFLite, where further activation reduction has little impact once the model weights are already highly compact [6].

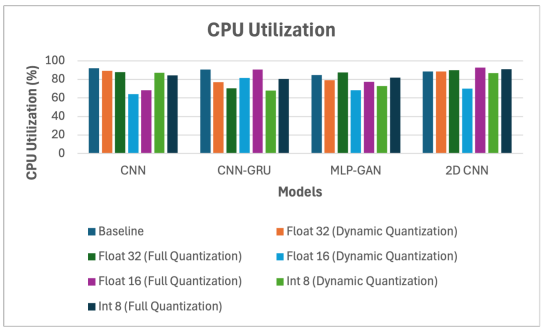


Figure 9: CPU Utilisation Comparison

Unlike the relationship shown between the model memory overhead and quantization, the CPU utilisation of both the quantized and baseline models as seen in Figure 9 appear to be similar. The quantized versions of the models all achieve a lower CPU utilisation than their baseline counterparts, however unlike the memory

reduction, the reduction in CPU usage is not as pronounced. The quantized CNN models achieved the highest utilisation with an average of 86.45% for the 2D CNN models and 80.11% for the 1D CNN models respectively. The quantized CNN-GRU models achieved the second highest CPU utilisation with an average of 77.9% and the quantized GAN-MLP models achieved the lowest average CPU utilisation of 77.7%. Despite the CNN architecture being more lightweight than the CNN-GRU, they achieved higher CPU utilisation rates because the standard convolutional layers map efficiently onto highly parallel CPU kernels, whereas the GRU layers in the CNN-GRU introduced sequential dependencies, limiting parallel execution. This is also true for the GAN-MLP where the MLP was required to wait for the discriminator to first classify the traffic before initiating multi classification. Thus the CNN-GRU and GAN-MLP achieved a lower average CPU usage despite the CNN-GRU possessing a more resource intensive architecture than the CNN.

From this experiment, it can be seen that the application of post training quantization both full and dynamic can effectively reduce the model's memory requirements while largely preserving multi classification performance. Additionally, it can be seen that CPU utilisation was less affected by quantization and more so by model architecture restricting parallelisation. Finally, fully quantizing the Int8 CNNs resulted in a reduction in multi classification performance illustrating the need for careful retraining when applying quantization.

6.3 Zero-Day Attack Detection Experiment

All retrained baseline models and their quantized versions were tested on their ability to generalise towards zero-day attacks. The quantized models, which did not suffer from class collapse, generalised as well as their baseline counterparts. In this section, we shall discuss the models as a whole.

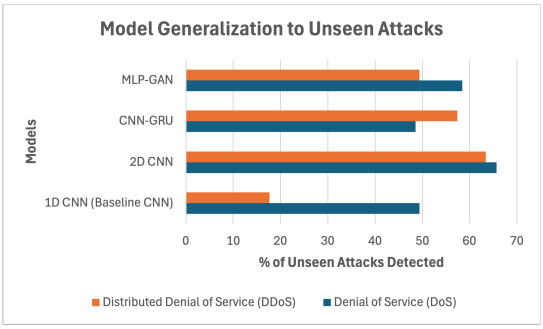


Figure 10: Comparison of Model Ability to Detect Zero-Day Attacks

As seen in Figure 10, all models are able to detect zero-day attacks. The GAN-MLP, CNN-GRU and 1D CNN all achieved relatively similar performance, flagging 58.42%, 48.55% and 49.39% of the unseen DoS attacks as malicious traffic. They also flagged 49.37%, 57.41% and 17.64% of the unseen DDoS attacks respectively.

The GAN-MLP is able to detect zero day attacks through the discriminator's ability to detect anomalous signatures in the network traffic. To serve as a safety net to reduce the number of false positives the MLP is able to classify traffic as normal traffic. This reduced the model's ability to detect zero-day attacks as the MLP classifier classified a large portion of the zero-day attacks as normal traffic.

The 2D CNN outperformed its 1D counterpart and flagged considerably more zero-day attacks as malicious with 65.71% of the DoS and 63.37% of the DDos attacks. The 2D CNN outperformed the 1D CNN as its 2 dimensional kernels are able to capture local cross-feature and spatial patterns within the input matrix of the network traffic. As such, the 2D CNN is able to better extract hierarchical and localised features resulting in stronger discrimination of spatially correlated features than the 1D CNN. This is shown in Figure 10 as the 1D CNN was unable to generalise towards detecting the zero-day DDos attacks, which have less consistent attack patterns, compared to its ability to generalise towards detecting the DoS attacks.

The CNN-GRU detected the least zero-day DoS attacks of all the models. This is because its convolutional layers extract localised spatial patterns more effectively for single-sample inputs [5]. This is counteracted by the GRU's sequential processing which tends to smooth or suppress attack signature features which may indicate novel attacks [5]. That being said, the GRU was able to locate the temporal features and as such, was able to generalise better towards detecting the zero-day DDoS attacks.

The results of this experiment demonstrate that while all models were able to detect zero-day attacks to some extent, the 2D CNN achieved the best performance. The results reveal limitations in both the CNN-GRU and GAN-MLP architectures. The CNN-GRU's performance suffered due to its GRU component suppressing the attack signature features and the GAN-MLP IDS performance suffered due to the MLP reclassifying the suspicious traffic as benign. They also highlight that 2D CNNs prove to be a more suitable classifier than their 1D counterpart in terms of detecting novel attack patterns in IDS applications.

7 DISCUSSION

The experiment results show that in regards to multi classification performance, both the 1D and 2D CNNs as well as the CNN-GRU perform well achieving an accuracy of 95%, 96% and 99% respectively. However, the GAN-MLP model underperformed only achieving an accuracy of 94%. This can be attributed to the lightweight architecture of the GAN and MLP as the MLP proposed by Rosay et al. [27] which achieved an accuracy of 99%, was too large for real world deployment. The CNN-GRU aligned with the prior work of Cao et al. [5], while the 1D CNN demonstrated that lightweight CNN IDS implementations remain feasible despite a minor performance trade-off.

Experiment 2 demonstrated that post-training quantization is a viable measure to reduce the models' memory requirements. All models maintained baseline classification performance post-quantization, however the CNN models showed signs of class collapse at Int8. Importantly, no quantized model exceeded an RSS of 1 MB, with

the 2D CNN peaking at 0.8 MB and the GAN-MLP reaching as low as 0.2 MB.

Unlike memory overhead, quantization did not have as pronounced an effect on the model's CPU utilisation which was rather driven by architectural constraints. This can be attributed to TensorFlow's multithreading behaviour, which generates spikes in CPU demand during inference [16]. The CNN layers mapped efficiently to parallel kernels, while the CNN-GRU and GAN-MLP incurred sequential dependencies lowering CPU utilisation. This suggests that further optimisation for CPU efficiency may require architectural modifications rather than post-training quantization.

While the models achieved similar CPU utilisation, model performance varied when faced with zero-day attacks. All models demonstrated some ability to generalise, however the 2D CNN was the most effective, detecting 63.3% of the DDoS and 65.7% of the DoS attacks. This aligns with Imrana et al. [15], who also developed a 2D CNN for zero-day detection.

By contrast, the 1D CNN generalised less effectively, particularly towards the DDoS attacks, which tend to exhibit more subdued irregular traffic patterns compared to DoS attacks. This highlights the advantages of 2D convolutional kernels which are able to better capture hierarchical and localised spatial correlations than the kernels of their 1D counterparts. The limitations of the 1D CNN were most evident with the DDoS attacks, where it detected only 17.6% of the zero-day attacks.

A limitation was revealed in the GAN-MLP model's architecture. While the discriminator aligned with Park et al. [21], detecting zero-day attacks, the auxiliary MLP multi-classifier frequently reclassified these as normal traffic, reducing zero-day detection to 48.7% for DoS and 42.6% for DDoS. However, under multi-classification performance, the MLP was hindered by the discriminator, which allowed attacks to slip through. These results highlight that unlike the CNN-GRU, whose components complemented each other by covering their weaknesses, the GAN-MLP's components only exacerbated each other's shortcomings. This demonstrates the importance of model selection when designing hybrid architectures.

Finally, the CNN-GRU performed well when detecting the DDoS attacks, detecting 57.4% of the attacks by leveraging the temporal pattern recognition of its GRU units. However, as also noted by Cao et al. [5], these units seemed to smooth and thus suppress the attack signature features of non-time based attacks resulting in the model performing the worst when generalising to DoS attacks only detecting 48.5% of the attacks.

8 CONCLUSION

The LAIDS project aims to develop a Lightweight AI-based IDS (LAIDS) to fill the security gap that is present in resource-constrained networks. As such this study proposed and evaluated four LAIDSs: a 1D CNN, a 2D CNN, a CNN-GRU, and a GAN-MLP. The models were evaluated on their baseline multi classification performance, post-training quantization resource usage, and generalisation towards zero-day attacks. The results highlight the trade-offs between model multi classification performance and model resource efficiency.

In terms of baseline performance, the CNN-GRU was the best performing model overall, achieving 99% accuracy, precision, recall, and F1 score, while also achieving an FPR of 0.003%. Additionally, the model generalised reasonably well towards zero-day. When quantized, its complex architecture resulted in the largest memory requirements of all the proposed models. Although the CNN-GRU leveraged spatial and temporal feature extraction effectively, its higher memory footprint highlights the trade-off between detection strength and lightweight deployment. That being said, with a quantized footprint of 0.7 MB, the model remains deployable in resource-constrained environments. The final decision on model deployment ultimately depends on the needs of the network administrator. While the CNN-GRU provides the strongest multi classification performance, it was the least memory efficient and did not generalise towards zero-day attacks as well as the 2D CNN. Conversely, the GAN-MLP, while the most lightweight with only 1.1 MB of memory overhead, was hindered by its discriminator, which often allowed malicious traffic to slip through and weakened the strong performance of the auxiliary MLP classifier. This imbalance between the two components ultimately reduced its performance.

Overall, the 1D CNN, 2D CNN, and CNN-GRU all represent promising lightweight IDS solutions which are feasible for real-world deployment. Post-training quantization was found to reduce memory usage dramatically, such as lowering the memory overhead of the 1D CNN from 64 MB to 0.4 MB, without major losses in accuracy. Ultimately, the optimal model selection depends on the specific deployment scenario's needs as each LAIDS demonstrates distinct strengths and limitations. These contributions demonstrate that optimised lightweight IDSs can be designed to strike an effective balance between detection performance and resource efficiency, making them viable defences for resource-constrained environments where traditional IDSs fall short

9 LIMITATIONS & FUTURE WORK

This study highlights several opportunities for further research. One such area for exploration is running these models in a resource-constrained environment such as a Raspberry Pi rather than the emulated environment used in this study. This will allow for a more accurate measurement of the models' resource requirements and the effect on overall device performance. Furthermore, measuring the resident set size of the models proved suboptimal once they became extremely lightweight, as the TensorFlow Lite interpreter overhead dominated the recorded memory usage rather than the model itself. An issue which running the model in a low resource environment would solve. Finally, expanding the models to handle a broader range of attack types would improve their robustness and enhance applicability to real-world intrusion detection scenarios, ultimately aiding in network security.

REFERENCES

[1] Moorthy Agoramoorthy, Ahamed Ali, D. Sujatha, Michael Raj. T F, and G. Ramesh. 2023. An Analysis of Signature-Based Components in Hybrid Intrusion Detection Systems. In *2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS)*. 1–5. https://doi.org/10.1109/ICCEBS58601.2023.10449209

[2] Tarek Ali, Amna Eleyan, Tarek Bejaoui, and Mohammed Al-Khalidi. 2024. Lightweight Intrusion Detection System with GAN-Based Knowledge Distillation. In *2024 International Conference on Smart Applications, Communications and Networking (SmartNets)*. 1–7. https://doi.org/10.1109/SmartNets61466.2024.10577682

[3] Muhammad Arsalan, Muhammad Mubeen, Muhammad Bilal, and Saadullah Farooq Abbasi. 2024. 1D-CNN-IDS: 1D CNN-based intrusion detection system for IIoT. In *2024 29th International Conference on Automation and Computing (ICAC)*. IEEE, 1–4.

[4] B. B. Borisenko, S. D. Erokhin, A. S. Fadeev, and I. D. Martishin. 2021. Intrusion Detection Using Multilayer Perceptron and Neural Networks with Long Short-Term Memory. In *2021 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*. 1–6. https://doi.org/10.1109/SYNCHROINFO51390.2021.9488416

[5] Bo Cao, Chenghai Li, Yafei Song, Yueyi Qin, and Chen Chen. 2022. Network intrusion detection model based on CNN and GRU. *Applied Sciences* 12, 9 (2022), 4184.

[6] Matteo Carnelos, Francesco Pasti, and Nicola Bellotto. 2025. MicroFlow: An Efficient Rust-Based Inference Engine for TinyML. *Internet of Things* 30 (2025), 101498.

[7] Swathi Ch and Suresh Babu Kare. 2024. A Comprehensive Analysis of Network Intrusion Detection in Internet of Things and Wireless Networks. In *2024 International Conference on Data Science and Network Security (ICDSNS)*. 01–05. https://doi.org/10.1109/ICDSNS62112.2024.10691047

[8] Wang Chengjun. 2009. Resident set management and realize. In *2009 Second International Workshop on Computer Science and Engineering*, Vol. 1. IEEE, 413–416.

[9] Sarra Cherfi, Ali Lemouari, and Ammar Boulaiche. 2025. Mlp-based intrusion detection for securing iot networks. *Journal of Network and Systems Management* 33, 1 (2025), 20.

[10] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of machine learning and systems* 3 (2021), 800–811.

[11] Peter I Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[12] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. Ieee, 1322–1328.

[13] J. Henriques, F. Caldeira, T. Cruz, and P. Simões. 2024. A survey on forensics and compliance auditing for critical infrastructure protection. *IEEE Access* 12 (2024), 2409–2444. https://doi.org/10.1109/ACCESS.2023.3348552

[14] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. 2021. Accurate post training quantization with small calibration sets. In *International conference on machine learning*. PMLR, 4466–4475.

[15] Yakubu Imrana, Yanping Xiang, Liaqat Ali, Adeeb Noor, Kwabena Sarpong, and Muhammed Amin Abdullah. 2024. CNN-GRU-FF: a double-layer feature fusion-based network intrusion detection system using convolutional neural network and gated recurrent units. *Complex & Intelligent Systems* 10, 3 (2024), 3353–3370.

[16] Yuriy Kochura, Sergii Stirenko, Oleg Alienin, Michail Novotarskiy, and Yuri Gordienko. 2017. Performance analysis of open source machine learning frameworks for various parameters in single-threaded and multi-threaded modes. In *Conference on computer science and information technologies*. Springer, 243–256.

[17] Mohammed A. Mahdi. 2024. Secure and Efficient IoT Networks: An AI and ML-based Intrusion Detection System. In *2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)*. 1–6. https://doi.org/10.1109/AIIoT58432.2024.10574789

[18] Stamatis Mastromichalakis. 2020. ALReLU: A different approach on Leaky ReLU activation function to improve Neural Networks Performance. *arXiv preprint arXiv:2012.07564* (2020).

[19] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089* (2018).

[20] Uneneibotejit Otokwala, Andrei Petrovski, and Harsha Kalutarage. 2024. Optimized common features selection and deep-autoencoder (OCFSDA) for lightweight intrusion detection in Internet of things. *International Journal of Information Security* 23, 4 (2024), 2559–2581.

[21] Cheolhee Park, Jonghoon Lee, Youngsoo Kim, Jong-Geun Park, Hyunjin Kim, and Dowon Hong. 2022. An enhanced AI-based network intrusion detection system using generative adversarial networks. *IEEE Internet of Things Journal* 10, 3 (2022), 2330–2345.

[22] Cheolhee Park, Jonghoon Lee, Youngsoo Kim, Jong-Geun Park, Hyunjin Kim, and Dowon Hong. 2023. An Enhanced AI-Based Network Intrusion Detection System Using Generative Adversarial Networks. *IEEE Internet of Things Journal* 10, 3 (2023), 2330–2345. https://doi.org/10.1109/JIOT.2022.3211346

[23] ND Patel, V Sudarsan Rao, and Ajeet Singh. 2024. QDNN-IDS: Quantized Deep Neural Network based Computational Strategy for Intrusion Detection in IoT. In *2024 IEEE Silchar Subsection Conference (SILCON 2024)*. IEEE, 1–7.

[24] Vinh Pham, Eunil Seo, and Tai-Myoung Chung. 2020. Lightweight Convolutional Neural Network Based Intrusion Detection System. *J. Commun.* 15, 11 (2020), 808–817.

[25] Vignesh Reddy, Sunitha R, M. Anusha, S Chaitra, and Abhilasha P Kumar. 2024. Artificial Intelligence Based Intrusion Detection Systems. *2024 4th International Conference on Mobile Networks and Wireless Communications (ICMNWC)* (12 2024), 1–6. https://doi.org/10.1109/icmnwc63764.2024.10872055

[26] Vignesh Reddy, Sunitha R, M. Anusha, S Chaitra, and Abhilasha P Kumar. 2024. Artificial Intelligence Based Intrusion Detection Systems. In *2024 4th International Conference on Mobile Networks and Wireless Communications (ICMNWC)*. 1–6. https://doi.org/10.1109/ICMNWC63764.2024.10872055

[27] Arnaud Rosay, Florent Carlier, and Pascal Leroux. 2019. MLP4NIDS: An efficient MLP-based network intrusion detection for CICIDS2017 dataset. In *International Conference on Machine Learning for Networking*. Springer, 240–254.

[28] Md Hasan Shahriar, Nur Imtiazul Haque, Mohammad Ashiqur Rahman, and Miguel Alonso. 2020. G-ids: Generative adversarial networks assisted intrusion detection system. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 376–385.

[29] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* 1, 2018 (2018), 108–116.

[30] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* 1, 2018 (2018), 108–116.

[31] Farhad Mortezapour Shiri, Thinagaran Perumal, Norwati Mustapha, and Raihani Mohamed. 2023. A comprehensive overview and comparative analysis on deep learning models: CNN, RNN, LSTM, GRU. *arXiv preprint arXiv:2305.17473* (2023).

[32] Shachar Siboni, Asaf Shabtai, and Yuval Elovici. 2018. Leaking Data from Enterprise Networks Using a Compromised Smartwatch Device. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18)*. https://dl.acm.org/doi/10.1145/3167132.3167214

[33] Bin Sun and Yu Zhao. 2024. TinyNIDS: CNN-Based Network Intrusion Detection System on TinyML Models in 6G Environments. *Internet Technology Letters* (2024), e629.

[34] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[35] Miracle Udurume, Vladimir Shakhov, and Insoo Koo. 2024. Comparative Evaluation of Network-Based Intrusion Detection: Deep Learning vs Traditional Machine Learning Approach. In *2024 Fifteenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 520–525.

[36] Patrick Vanin, Thomas Newe, Lubna Luxmi Dhirani, Eoin O'Connell, Donna O'Shea, Brian Lee, and Muzaffar Rao. 2022. A Study of Network Intrusion Detection Systems Using Artificial Intelligence/Machine Learning. *Applied Sciences* 12, 22 (2022), 11752. https://doi.org/10.3390/app122211752

[37] Zhendong Wang, Hui Chen, Shuxin Yang, Xiao Luo, Dahai Li, and Junling Wang. 2023. A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization. *PeerJ Computer Science* 9 (2023), e1569.

[38] Zhenghong Wang, Sijie Ruan, Tianqiang Huang, Haoyi Zhou, Shanghang Zhang, Yi Wang, Leye Wang, Zhou Huang, and Yu Liu. 2024. A lightweight multi-layer perceptron for efficient multivariate time series forecasting. *Knowledge-Based Systems* 288 (2024), 111463. https://doi.org/10.1016/j.knosys.2024.111463

[39] Zhendong Wang, Yong Zeng, Yaodi Liu, and Dahai Li. 2021. Deep Belief Network Integrating Improved Kernel-Based Extreme Learning Machine for Network Intrusion Detection. *IEEE Access* 9 (2021), 16062–16091. https://doi.org/10.1109/ACCESS.2021.3051074

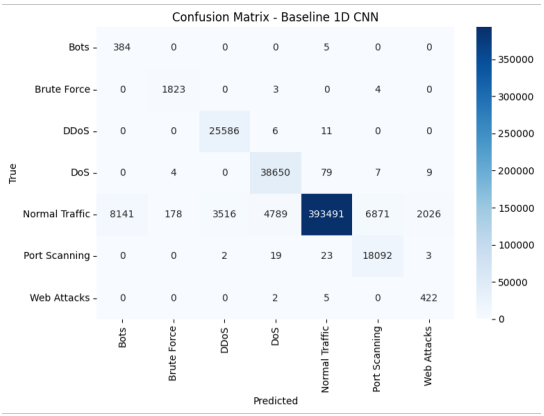
[40] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. 2019. Quantization Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

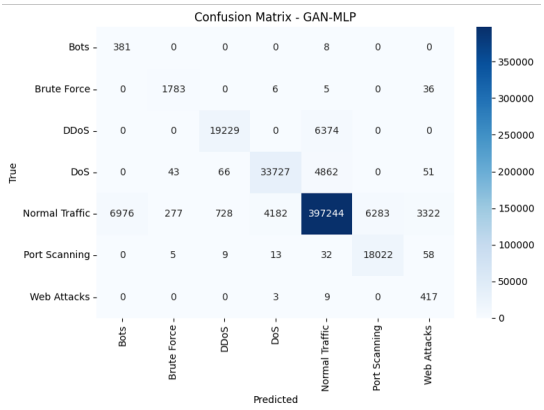
[41] S Zargar. 2021. Introduction to sequence learning models: RNN, LSTM, GRU. *Department of Mechanical and Aerospace Engineering, North Carolina State University* (2021).

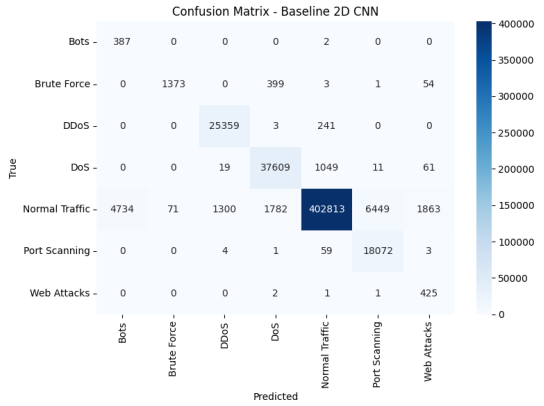
APPENDIX

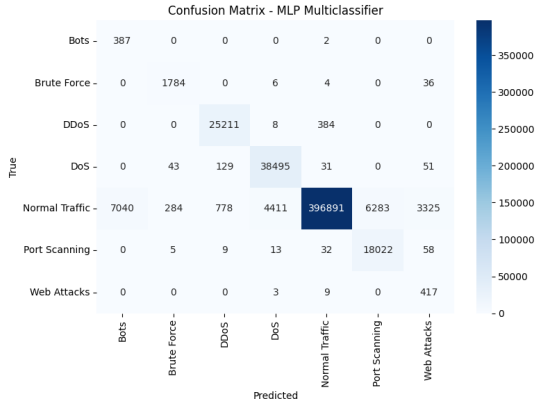
A CONFUSION MATRICES OF BASELINE MODELS

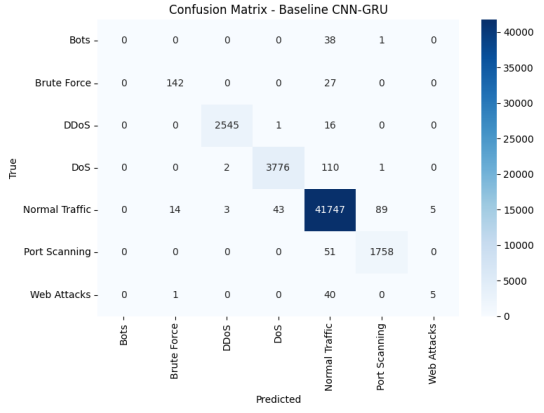
Below are the confusion matrices further detailing model performance.

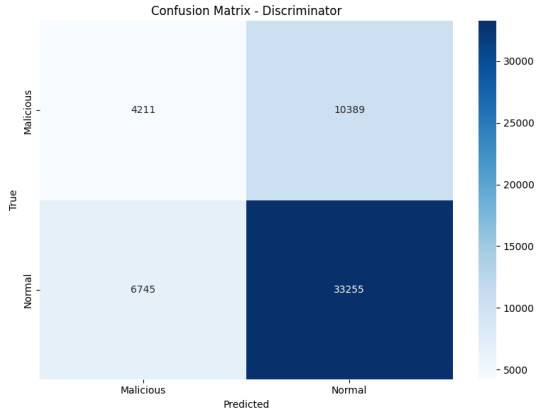












B DETAILED PERFORMANCE OF QUANTIZED MODELS

Table 2: Comparison of Quantization Performance Across CNN-GRU, GAN-MLP, 1D CNN, and 2D CNN Models

Model	**Model Weights**	Accuracy (%)	Precision (%)	F1 Score	Recall (%)	False Positive Rate (%)	Memory Usage (MB)	Average CPU Utilisation (%)
CNN-GRU	Float 32 (Dynamic)	99	99	99	99	0.003	0.7	76.8
	Float 32 (Full)	99	99	99	99	0.005	0.6	70.5
	Float 16 (Dynamic)	99	99	99	99	0.003	0.7	81.6
	Float 16 (Full)	99	99	99	99	0.005	0.6	90.4
	Int 8 (Dynamic)	99	99	99	99	0.003	0.6	67.8
	Int 8 (Full)	0.1	0.1	0.1	0.8	0.000	0.6	80.3
GAN-MLP	Float 32 (Dynamic)	93	95	94	93	5.2	0.4	79.2
	Float 32 (Full)	93	95	94	93	5.2	0.3	87.3
	Float 16 (Dynamic)	93	95	94	93	5.2	0.3	68.1
	Float 16 (Full)	93	95	94	93	5.2	0.2	77.2
	Int 8 (Dynamic)	94	96	95	94	4.7	0.3	72.7
	Int 8 (Full)	83	69	75	83	0	0.3	81.8
1D CNN	Float 32 (Dynamic)	95	98	96	95	6.1	0.4	89.3
	Float 32 (Full)	95	98	96	95	6.1	0.3	87.8
	Float 16 (Dynamic)	95	98	96	95	6.1	0.4	64.1
	Float 16 (Full)	95	98	96	95	6.1	0.3	68.1
	Int 8 (Dynamic)	95	98	96	95	6.1	0.3	87.2
	Int 8 (Full)	83	73	75	83	0.0	0.4	84.2
2D CNN	Float 32 (Dynamic)	96	98	97	96	4.0	0.8	88.3
	Float 32 (Full)	97	98	97	97	4.3	0.3	89.7
	Float 16 (Dynamic)	97	98	97	97	4.0	0.2	70.1
	Float 16 (Full)	97	98	97	97	4.3	0.3	92.8
	Int 8 (Dynamic)	96	98	97	96	3.7	0.2	86.8
	Int 8 (Full)	93	93	93	93	0.0	0.4	91.0