A Lightweight AI-Based Intrusion Detection System (IDS) For Resource-Constrained Networks

Claire Campbell
The University of Cape Town
Cape Town, South Africa
cmpcla004@myuct.ac.za

ABSTRACT

Many Internet of Things (IoT) based edge devices remain constrained in compute, memory and energy and thus, their deployment of conventional, resource-intensive Artificial Intelligence (AI) based Intrusion Detection Systems is limited. This study investigates whether lightweight deep models can meet device constraints without sacrificing detection, and how post-training quantisation affects that trade-off. A Baseline Convolution Neural Network (CNN), a hybrid CNN-Long Short Term Memory (CNN-LSTM), and a Deep Belief Network (DBN) are evaluated on CICIDS2017 under edge-realistic conditions (batch size=1), reporting accuracy and F1 alongside latency, throughput, and memory footprint. Three post-training precisions, Float16, INT8 Dynamic (weight-only), and full INT8, are applied and zero-day generalisation is assessed by holding out one attack class and operating at a fixed false-positive rate (FPR) on Normal Traffic. Results demonstrate that Float16 presents as a safe, overall quantisation default. INT8 Dynamic achieves the largest reductions and, CNN-LSTM, improves speed with no accuracy loss, thus making it the most-balanced candidate. While INT8 degrades performance and requires Quantisation-Aware Training. Generalisation was strongly dependent on the false-positive rate selected. At $FPR = 30(Bots)$, the CNN is most sensitive and at $FPR = 0.15(DoS)$, the DBN is most sensitive. Overall, the CNN-LSTM INT8 Dynamic offers a strong balance between accuracy, latency, and size.

KEYWORDS

Intrusion Detection Systems, Artificial Intelligence

1 INTRODUCTION AND MOTIVATION

With billions of IoT devices now embedded in environments ranging from smart homes to industrial systems, their limited computational capacity and constant data exchange underscore the need for lightweight yet effective security mechanisms [5, 24]. Dynamic topologies and heterogeneous hardware further expose security vulnerabilities [6]. Attacks such as botnets, denial-of-service (DoS), and distributed denial-of-service (DDoS) have been shown to compromise both privacy and availability, exacerbated by weak or outdated security protocols [34, 35]. Lightweight, effective protection for resource-constrained environments is therefore a pressing need.

Intrusion Detection Systems (IDSs) monitor network traffic for malicious activity and remain a central network defence [46]. Signature-based IDSs are efficient against known threats but fail on novel attacks, while anomaly-based IDSs can flag unknown attacks but often suffer high false-positive rates and scalability limits [37, 39].

The success of artificial intelligence (AI) and machine learning (ML) has driven their use in IDSs to capture patterns that traditional methods do not [25]. AI-based IDSs can automatically learn useful features from raw traffic to achieve strong accuracy, even with imbalanced training data [5, 19, 25, 41]. However, AI implementations are often computationally expensive for low-resource devices where memory, latency, and energy constraints dominate [22]. This tension between accuracy and efficiency motivates lightweight AI-IDS designs that operate in real time without overwhelming device resources [3].

This study investigates whether compact AI models can meet resource constraints while maintaining sensitivity to both seen and unseen attacks. Addressing this gap is vital for safeguarding the reliability and resilience of ecosystems, including smart cities, healthcare, and industrial control systems [5, 24]. The paper compares three lightweight IDS models to which post-training quantisation (store values with fewer bits) is applied to save memory and speed up inference. Finally, the benchmarked models will be tested against zero-day generalisation by omitting one attack from training and testing at a fixed false-positive rate.

1.1 Motivation

1.1.1 Problem Statement. Resource-constrained environments, spanning networks, edge devices, and embedded systems, continue to be exposed to evolving attacks. This is due to limited compute, memory and energy budgets, thus addressing vulnerabilities in a resource-efficient manner has become a critical area of research. Existing IDS solutions, signature-based and anomaly-based, and AI-IDS, either fail to generalise to unseen attacks (zero-day attacks) or impose computational costs that exceed typical on-device resources [1, 9, 27]. Consequently, these systems remain vulnerable. Thus, there is a need for research into a lightweight, AI-driven IDS capable of zero-day attack detection in near-real time, while operating within strict device-level constraints.

1.1.2 Research Question. The study aims to develop a series of lightweight AI-IDSs suitable for deployment in a resource-constrained network as well as explore how constrained computational resources influence IDS performance in terms of accuracy, precision and false negative rate. To achieve these aims, the following research questions are explored in this paper:

Questions

(1) How can a Hybrid CNN-LSTM IDS and DBN IDS achieve resource efficiency, compared to a CNN IDS, while maintaining or improving detection performance?

(2) For a CNN, CNN-LSTM, and DBN IDS, which Post-Training Quantisation, between Float16, INT8 Dynamic, and INT8 Full, yields the best quality-latency-size trade-off relative its Float32 model?
(3) At fixed false-positive budgets on Normal Traffic, to what extent can a CNN IDS, CNN-LSTM IDS, and a DBN IDS generalise to unseen data to detect novel threat scenarios without retraining?

2 BACKGROUND AND RELATED WORK

2.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs) consist of multiple layers, each responsible for learning and extracting relevant features to enhance data classification [37]. Due to their ability to analyse high-dimensional data and recognise complex patterns, CNNs are particularly well-suited for network traffic analysis [32]. CNN algorithms can effectively distinguish between normal and abnormal data, addressing a key limitation faced by traditional anomaly-based IDS [10].

Both Xiao et al. [43] and Pham et al. [29] leverage convolutional architectures for intrusion detection, but with different problem formulations. Xiao et al. [43] propose CNN-IDS, a LeNet-5–based convolutional model in which traffic from KDD-CUP99 is normalised and reduced via Principal Component Analysis and Autoencoders, then reshaped into 2D feature maps suitable for CNN input. The network alternates between convolution and pooling blocks with batch normalisation and dropout, and a Softmax head predicts five classes (DoS, Probe, R2L, U2R, Normal). In contrast, Pham et al. [29] design a specifically lightweight CNN-based IDS that reconstructs flow detection as image classification by encoding short windows of PCAP traffic as grayscale images using packet length and arrival time. Their compact CNN comprises of four convolutional blocks with seven-way Softmax, implemented in Keras/TensorFlow with Adam and categorical cross-entropy. Viewed together, both studies exploit CNNs' ability to transform network flows into spatial feature maps, although using different encodings and architectural depth.

From an efficiency and deployment perspective, both papers allude toward speed but quantify it differently. Xiao et al. [43] report faster training and reduced classification time relative to DNN/RNN baselines. However, the study does not quantify resource constraints nor apply model compression such as quantisation. Thus, the CNN-IDS is better described as an efficient CNN than as a lightweight IDS. Pham et al. [29] report 95% F1 on CSE-CIC-IDS2018 and an average per-image detection time of 3.23 microseconds, and propose a two-stage deployment that routes known benign application traffic via a fast Payload Size Distribution classifier while forwarding unknown flows to the CNN detector. However, they do not report device-level latency or energy on constrained hardware, which limits direct comparison with deployment-focused lightweight IDS. Sun et al. [38] explicitly target TinyML by designing a lightweight CNN tailored for 5G/6G edge nodes. The study employs depth-wise convolutions, post-training quantisation, and pruning to meet low-latency budgets while maintaining low false-positive and negative rates. Although latency is reported, there is limited discussion on memory and energy consumption. Across these studies [29, 38, 43] compact CNNs are demonstrated as accurate and fast in principle, but reporting of on-device constraints is limited.

2.2 Convolutional Neural Network-Long Short Term Memory

While CNNs efficiently process large IoT datasets, their high computational cost can strain available resources [32]. To address this challenge, hybrid CNN architectures, where CNN layers are combined with other models, have been proposed to reduce computational load while maintaining optimal performance [20]. One such model often used with CNNs are Long Short-Term Memory (LSTM) models, a type of Recurrent Neural Network. LSTMs excel at mapping temporal features in sequential data, mitigating the vanishing gradient problem and proving effective in modelling network traffic flows for IDS [14, 33]. LSTMs are widely adopted in IDS for capturing long-range temporal dependencies, which can enhance detection accuracy [37].

Both Du et al. [11] and Jouhari & Guizani [19] adopt a hybrid design that pairs convolutional front-ends with sequence modelling to capture temporal structure in network flows. Du et al. [11] introduce NIDS-CNNLSTM with a two-layer CNN followed by a two-layer unidirectional LSTM. The CNN replaces fully connected layers with global average pooling to reduce parameters, and a Softmax head outputs class probabilities. Similarly, Jouhari & Guizani [19] use a lightweight 1D-CNN, one convolutional and pooling layer, whose feature sequence is consumed by a bidirectional LSTM to model dependencies in both directions. Du et al. [11] evaluates on benchmark datasets (KDD-CUP99, NSL-KDD, UNSW-NB15 and Jouhari & Guizani [19]) evaluates on UNSW-NB15. Both studies report strong accuracy in binary and multi-class settings. Jouhari & Guizani [19] use class-weighted loss to address imbalance.

From a deployment standpoint, however, both works emphasise accuracy over quantified efficiency. Du et al. [11] do not report parameter counts, memory footprint, or latency, while Jouhari & Guizani [19] provide PC-level inference time of 4 seconds but omit device-level memory, energy and detailed model size. This leaves deployment on constrained hardware unclear. Neither papers [11, 19] explore compression or reports on-device measurements, so although CNN–LSTM hybrids appear promising for IDS, their resource consumption and trade-offs require further validation.

2.3 Deep Belief Network

DBNs have also gained popularity in IDS research for their hierarchical feature learning and complex pattern detection [7, 21, 40, 45]. DBNs are generative deep neural networks composed of stacked Restricted Boltzmann Machines (RBMs) which are generative stochastic neural networks that learn a probability distribution over a hidden and visible layer [7]. The layered structure of DBNs is particularly useful for filtering out noise during feature extraction and classification in IDSs [7, 45].

Both Tan et al. [40] and Balakrishnan et al. [7] build on the common DBN framework of greedy layer-wise pre-training with stacked RBMs followed by back-propagation fine-tuning for classification. Tan et al. [40] add Particle Swarm Optimization (PSO) to select the number of hidden units per layer and report 92.44% accuracy on KDD-CUP99, arguing that DBNs capture high-dimensional, non-linear traffic patterns. In parallel, Balakrishnan et al. [7] couple RBM pre-training with feature normalisation and obtain strong per-attack scores for prevalent classes under a 60/30/10 split, while noting lower performance on minority classes. The two studies underscore DBN's appeal for representation learning in IDS. Tan et al. [40] achieves this via PSO tuning, and Balakrishnan et al. [7] via preprocessing and fine-tuning. However, Tan et al. [40] uses legacy dataset, specifically the KDD-CUP99, which limits conclusions about modern traffic.

From a lightweight deployment perspective, both works emphasise accuracy over quantified efficiency. Tan et al. [40] frame the method for Unmanned Aerial Vehicles network security but do not report parameter counts, memory footprint, latency, or compression. Likewise, Balakrishnan et al. [7] gesture toward IoT applicability yet omit device-level measurements and model size, despite the DBN pipeline's promising results on common attacks. Consequently, while the papers collectively motivate DBNs for IDS, they omit information on how DBNs scale on constrained CPUs, how sensitive they are to class imbalance in new datasets, and whether post-training compression or architecture simplification can deliver practical latency and memory usage.

3 MODEL DESIGN AND IMPLEMENTATION

This study evaluates three lightweight IDS models on CICIDS2017: a 1D CNN, a CNN–LSTM, and a DBN. A 2D CNN variant is also evaluated in the generalisation experiment in Section 4.5. This section specifies the dataset representation, preprocessing, and the model architectures only. All training protocol and benchmarking procedure are defined in Section 4.3 (Experimental Methodology: Benchmarking Models).

3.1 CICIDS2017 Dataset

The CICIDS2017 dataset was created by the Canadian Institute for Cybersecurity and captures multi-day, realistic network activity with both benign and attack traffic [13]. The dataset encompasses a spectrum of attacks for example, DoS, DDoS, SSH/FTP Brute Force, Web Attacks, Heartbleed, Infiltration, and Botnet across 86 engineered features [23]. Traffic was captured over five days (3-7 July 2017) and released as labelled flows with corresponding pcaps, organised into eight captures [28].
A class imbalance is present, with benign and certain majority attacks dominating samples, motivating explicit balancing techniques in the training pipeline [28, 39]. Older IDS datasets such as KDD99 and NSL-KDD are either dated, limited in diversity or volume making them unreliable for modern threats [36]. This motivates the preference for newer datasets [36].

This study adopts Anacleto's [4] preprocessed CICIDS2017 release accessed via Kaggle. It is derived from the original dataset [36]. The pipeline that Anacleto [4] followed involved merging the original multiple CSVs, removing duplicate rows and columns, converting infinite values to NaN and handling them alongside other missing values. Headers were normalised, and data-driven feature selection was performed by removing constant-value columns, pruning one feature from each pair with near-perfect correlation, and applying Kruskal-Wallis and Random-Forest-based selection to discard statistically irrelevant features. Additionally, similar attacks were grouped together and rare classes were removed to minimise overfitting and to improve generalisation.

Despite this prior cleaning by Anacleto [4], CICIDS2017 remains highly imbalanced, with Normal Traffic dominating. Adaptive Synthetic sampling (ADASYN) was subsequently applied only to the train split. ADASYN adaptively generates minority samples based on their distributions and on which patterns are more complex to learn [16]. A class-wise sampling strategy was used that synthesised minority samples up to approximately 65% of the capped majority size, with k=5 neighbours and a fixed random seed of 42 for reproducibility. No resampling was applied to the validation or test-sets to retain the natural class skew and serve as unbiased holdouts. The resulting class distributions for each split are shown in Figure 3.1.

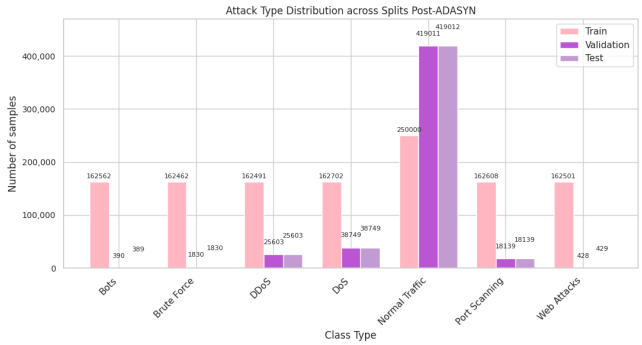

Figure 3.1: Class Distributions per Split after Preprocessing.

3.2 Key metrics

Standard classification and resource usage metrics are used to evaluate the IDS models.

3.2.1 Confusion Matrix. A confusion matrix compares true labels with predictions and summarises true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) [31]. Minimising false positives reduces false alerts [41], while minimising false negatives reduces missed attacks [44].

3.2.2 Accuracy. Overall accuracy measures the proportion of correct predictions. It reflects the overall correctness across normal and attack traffic by evaluating the proportion of correctly classified instances over the total number of cases [19].

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} [44] \quad (1)$$

3.2.3 Precision. Measures the proportion of true positive classifications relative to all instances classified as positive. It helps assess the IDS's ability to correctly identify actual threats [31].

$$\text{Precision} = \frac{TP}{TP + FP} [31] \quad (2)$$

3.2.4 Recall. Measures how well an IDS captures actual attacks by computing the proportion of detected intrusions out of all real intrusions in the dataset [31].

$$\text{Recall} = \frac{TP}{TP + FN} [44] \quad (3)$$

3.2.5 F1-score. Combines precision and recall into a single value via the harmonic mean, rewarding models that keep both high. It ranges from 0 to 1 and is high only when both precision and recall are high [19, 39]. Additionally per-class $F_{1,c}$ is aggregated as macro-F1 (unweighted mean) and weighted-F1 (support-weighted mean) to reflect class imbalance.

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} [44] \quad (4)$$

Accuracy is informative but can be misleading under class imbalance, and thus this research does not solely rely on it as a metric [19]. Recall is critical to minimise missed attacks (false negatives), while precision is vital to reduce false alarms and operational burden. F1 balances these two perspectives for a more nuanced view of detection quality [19, 39].

3.2.6 False Alarm Rate. When mapping to Normal versus Attack:

$$\text{FAR} = \frac{\text{FP on Normal}}{\text{Total Normal}} \times 100. \quad (5)$$

3.2.7 False Negative Rate. When mapping to Normal versus Attack:

$$\text{FNR} = \frac{\text{FN on Attacks}}{\text{Total Attacks}} \times 100. \quad (6)$$

3.2.8 Resource usage.

(1) *Storage* (MB): serialized model size.
(2) *Latency* (ms/sample): mean per-sample forward-pass time at batch size 1.
(3) *Throughput* (samples/s): processed samples per second.
(4) *CPU time* (ms/sample): user+system CPU time per sample.
(5) ΔRSS (MB): change in resident set size across an inference run (approximate incremental working set).

3.2.9 Zero-day detection at an operating point. The score s is calculated and if $s \geq threshold$, the sample is marked as an unknown attack. The threshold is chosen on validation to hit a target false–positive rate (FPR) on Normal Traffic.

$$\text{s} = 1 - p(Normal) \quad (7)$$

On test, true–positive and false–positive rates are reported:

$$\text{TPR} = \frac{\text{Unseen flagged as unknown}}{\text{Unseen total}} \quad (8)$$

$$\text{FPR} = \frac{\text{Normal flagged as unknown}}{\text{Normal total}} \quad (9)$$

3.3 Baseline CNN Model

CNNs are strong candidates for IDS because hierarchical convolutions extract local patterns from high-dimensional flow features while remaining lightweight [10, 32, 37]. This 1D CNN serves as the baseline against which the CNN–LSTM and DBN are compared.

After standardisation (see Section 3.1), each example is represented as L=52 per-flow features and fed as a 1D signal of shape $(L, 1)$ with integer-encoded labels.

Drawing from established designs for lightweight CNN-based Intrusion Detection Systems, the network consists of two convolutional blocks followed by a classifier head. This architecture is particularly effective as it captures local dependencies in the per-flow features, a method supported by prior work [32, 37]. Similar to Qazi et al. [30], the architecture of the baseline uses a 1D CNN.

Following the design principles of Qazi et al. [30], the first convolutional block is configured with 32 filters and a kernel size of 2, which has been shown to be effective for initial feature extraction on network data. This is followed by batch normalisation to normalise the outputs, before passing to the Max Pooling layer to reduce dimensions of the data [18]. The second convolutional block comprises a convolutional layer with 16 filters and a kernel size 3, again followed by Batch Normalisation and Max Pooling [30]. In the final model, Flatten is replaced with global average pooling, based on Sun et al. [38], which reduces parameters. The classifier then uses a Dense layer with 64 units (ReLU), Dropout(0.3) for regularisation, and a final Softmax for multi-class classification. The overall design uses common lightweight CNN IDS patterns reported to perform well on network traffic [32, 37, 38]. Figure 3.2 depicts the CNN architecture.

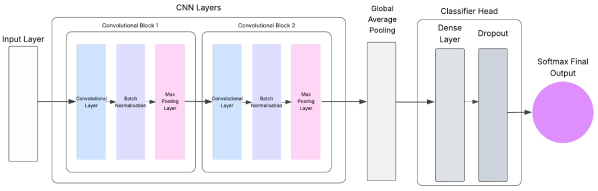

Figure 3.2: CNN Architecture.

The model's robust architecture is complemented by a systematic training and tuning regimen. Similar to Aljumah et al. [2], the model is trained with Adam and sparse categorical cross-entropy. Mini-batches of size 64 are used for 20 epochs, with early stopping monitoring validation loss with patience of 6 and restoring the best

weights to limit overfitting and unnecessary computation. The final deployed configuration uses an Adam learning rate of 3×10^{-4}. Further detail on the data splits and training are explored in Section 4.3 and the results and discussion of the model are presented in Section 5.1.

Hyperparameters are an important consideration as they control training behaviour and significantly impact performance [42]. Bayesian Optimisation (KerasTuner) was selected for CNN tuning given the time and resource constraints which limited the exhaustive evaluation of all possible configurations [12]. Bayesian Optimisation sets a prior over the optimisation function and updates its posterior using information gathered from previous samples [42].

The search covered the convolutional filters and kernel sizes, Dense units and L2, the two Dropout rates, and the Adam learning rate. The objective was to find the optimal combination of these parameters that maximized the model's validation accuracy. To ensure that the model did not become too specialised in the training data, early stopping was implemented to prevent overfitting. This systematic approach allowed for an efficient exploration of the parameter space, leading to the selection of the best-performing configuration for the final model. Table 3.1 displays the results of CNN tuning. Following the search, three small, post-tuning adjustments were applied for deployability and quantisation. GlobalAveragePooling1D was used in place of Flatten, a second dropout of 0.3 was used, and finally a reduced learning rate of 3×10^{-4}.

3.3.1 2D CNN. Motivated by prior reports that 2D kernels capture cross-feature structure and can aid zero-day detection [17, 29, 43], Experiment 4.5 extends the 1D CNN to a 2D variant. The model reshapes each flow to $7 \times 8 \times 1$. The network applies two Conv2D layers: 32 filters with 2×2 kernels and 16 filters with 3×3 kernels (ReLU), followed by Batch Normalisation, a 2×2 Max Pooling, Global Average Pooling, a 64–unit ReLU Dense layer with Dropout $p = 0.3$, and a Softmax output. Training uses Adam (learning rate 3×10^{-4}) with sparse categorical cross–entropy, preprocessing and class–imbalance handling match the 1D CNN.

Layer	Hyperparameter Choices	Best Parameter
Conv Block 1	Filters: {8, 16, 32, 64, 128}	32
	Kernel Size: {2, 3, 5}	2
Conv Block 2	Filters: {8, 16, 32, 64, 128}	16
	Kernel Size: {2, 3, 5}	3
Dense	Units: {8, 12, 24, 64}	64
	L2 Regularizer: $\{0.0, 10^{-3}, 10^{-4}, 10^{-2}\}$	0.0
Dropout	Dropout Rate (1): {0.0, 0.2, 0.25, 0.3, 0.5}	0.0
	Dropout Rate (2): {0.0, 0.2, 0.25, 0.3, 0.5}	0.5
Optimizer	Learning Rate: $\{10^{-3}, 5 \times 10^{-3}, 10^{-2}\}$	5×10^{-3}

Table 3.1: Summary of Hyperparameter Tuning Results via Bayesian Optimization for Baseline CNN

3.4 CNN-LSTM Model

Following prior IDS hybrids that pair convolutional encoders with recurrent temporal modelling [11, 15, 19], a compact CNN–LSTM is adopted. A convolutional front-end first extracts high-level features at each time-step, after which an LSTM models temporal structure across the sequence [15]. Non-overlapping windows of $T = 10$ from the standardised per-flow features. Each window is labelled by the last time step, and inputs are shaped $(T, L, 1)$.

The data passes through two TimeDistributed 1D convolutional blocks that learn local spatial patterns. The first block consists of 16 filters (kernel size 3) followed by Max Pooling to preserve the dominant features, and then Batch Normalisation [15]. The second block consists of 32 filters (kernel size 3) with Max Pooling. Per-step feature maps are flattened and passed to a single LSTM layer, consisting of 32 units, that captures temporal dependencies across the windows. This is followed by a Dropout layer to regularise sequence representation and prevent overfitting [15]. A lightweight classifier head applies a Dense(12, ReLu) and a Softmax output to produce class probabilities. Sparse categorical cross-entropy with Adam (3×10^{-4}) is used for optimisation. Accuracy is tracked during training, and macro-F1 is monitored via validation callback to address class imbalance [19]. Figure 3.3 depicts the CNN-LSTM architecture. Further information on data splits and training are mentioned in Section 4.3 and the results of the model are presented and discussed in Section 5.

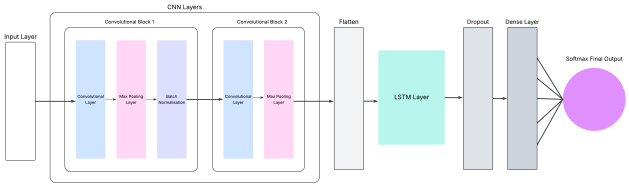

Figure 3.3: CNN–LSTM Architecture.

Similar to the Baseline CNN, Bayesian Optimisation was also selected to explore filter and kernel sizes, LSTM width, dropout, an optional dense width, and the learning rate, with early stopping on validation loss to prevent overfitting. Table 3.2 summarises the results of tuning. After comparing accuracy, recall and F1 scores, the final configuration was selected with $Conv1 = 16$ and $Conv1 = 32$, both with kernel size 3. This selected model outperformed the tuning result's leaner alternative.

Layer	Hyperparameter Choices	Best Parameter
Conv Block 1	Filters: {8, 16, 32, 64}	8
	Kernel Size: {3, 5}	3
Conv Block 2	Filters: {16, 32, 64, 128}	16
	Kernel Size: {3, 5}	5
LSTM	Units: {32, 64, 128}	32
	Dropout Rate: {0.2, 0.3, 0.5}	0.3
Optional Dense	Use Extra Dense: Boolean	False
Optimizer	Learning Rate: $\{10^{-4}, 3 \times 10^{-4}, 10^{-3}\}$	3×10^{-4}

Table 3.2: Summary of Hyperparameter Tuning Results via Bayesian Optimization for CNN-LSTM

3.5 Deep Belief Network Model

Following prior IDS work that leverages Deep Belief Networks (DBNs) for hierarchical feature learning on noisy, high-dimensional

traffic features, a compact DBN is adopted for non-sequential, per-flow inputs [7, 21, 26, 40, 45]. A DBN is formed by stacking Restricted Boltzmann Machines (RBMs) and training them greedily layer-by-layer to initialise a deep model before supervised fine-tuning, which has been shown to yield effective representations for IDS tasks [7, 40, 45].

Two RBM layers are pre-trained on the standardised training features, mapping the visible layer d to 64 hidden units and then to 32 hidden units. Following the work of Nivaashini et al. [26], the DBN follows a two-stage training process. The decreasing layered structure aligns with the work of Belarbi et al. [8]. Each RBM is trained with contrastive divergence to minimise mean sample error, using a learning rate of 10^{-2}, momentum of 0.5, weight decay of 10^{-4}, and batch size 256 for 30 epochs [8, 26]. The stacked RBM parameters are then used to initialise a lightweight feed-forward classifier which consists of a ReLU and Dropout 0.3 block feeding a 128-unit dense layer, followed by ReLU, Dropout 0.3, and a final linear C, where C is the number of classes. Finally Softmax is applied at evaluation to obtain class probabilities.

After greedy pre-training, the full network is fine-tuned end-to-end with cross-entropy and Adam (learning rate 10^{-3}, weight decay 10^{-4}), using batch sizes of 512 on training and 1024 for validation and testing. Early stopping monitors validation macro-F1 ($patience = 6$) which tracks the best run, and training runs up to 40 epochs. Prior literature reports that such greedy layer-wise initialisation improves convergence and generalisation in IDS contexts, motivating the design choice for a lightweight DBN on tabular network features [7, 40, 45]. Figure 3.4 depicts the DBN architecture.

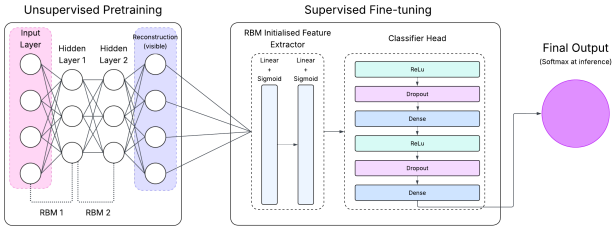

Figure 3.4: DBN Architecture.

The DBN operates on flat per-flow features. Due to the manual specification of a compact DBN and to keep the model lightweight, an automated hyperparameter search was not executed but rather settings were fixed in advance based on brief pilot experiments. Further details on data splits and training are mentioned in Section 4.3 and the results of the model are presented and discussed in Section 5.

4 EXPERIMENTAL METHODOLOGY

The experimental element of this study is necessary to identify a lightweight AI-IDS model which balances both resource usage as well as performance in terms of precision, recall, F1, and overall generalisation. This section specifies the environment under which all experiments were run, the ethical and legal issues associated with experimentation, and the methodology followed for each experiment. The results achieved from the three experiments described below, can be found in Section 5.

4.1 Computational Environment and Software

All experimental setups and execution were run in Google Colab, and in some cases the NVIDIA T4 GPU, and the High-RAM runtime for hyperparameter tuning. Data handling used pandas and NumPy, while preprocessing employed scikit-learn (StandardScaler, LabelEncoder, StratifiedShuffleSplit) and imbalanced-learn for ADASYN. The CNN and CNN-LSTM were implemented in TensorFlow/Keras, trained with Adam and sparse categorical cross-entropy with early stopping, and quantised via the TensorFlow Lite converter. The DBN was implemented in PyTorch with custom RBM layers to initialise a DBNClassifier. Fine-tuning used Adam with CrossEntropyLoss and DataLoader batching, and PyTorch dynamic quantisation. Metrics were computed with Scikit-learn, and visualisations used Matplotlib and Seaborn. Artifacts were persisted as Keras (.keras) models, torch.save checkpoints, joblib pickles (scaler/encoder), and NumPy .npz/.npy files.

4.2 Ethical, Professional and Legal Issues

4.2.1 Legal Considerations. The study avoids the use of human data in model development and experimentation to avoid any legality issues. Compliance with privacy regulations, such as the Protection of Personal Information Act (POPIA) is ensured by using the publicly available CICIDS2017 dataset to avoid any legal concerns with processing sensitive information. All outputs (code, models, documentation) are open-source to promote transparency and lawful distribution.

4.2.2 Ethics clearance. No ethical clearance was required, as this study does not involve human participants or sensitive personal data. The primary ethical consideration, dataset usage, is addressed by using the thoroughly anonymised and ethically sound, CICIDS-2017 dataset which has been used widely in research. All data processing occurs within isolated local machines, preventing harm to real-world systems. The AI development will not involve profiling nor will it make generalisations based on the origin of network traffic within the anonymised dataset. By using only publicly available, anonymised data, the AI-IDS operates without interacting with real individuals or live systems, mitigating the common ethical concerns associated with AI.

4.3 Benchmarking Models

The study focuses on benchmarking three lightweight AI-IDS models on CICIDS2017 to establish a performance baseline, to answer Research Question 1. This experiment benchmarks three distinct architectures on CICIDS2017: a CNN, a CNN-LSTM, and a DBN. The objective is to establish a performance baseline. Flow-level records are worked with as described in Section 3.1 and the architectures of each model are specified in Section 3. Model-specific tuning strategies and final configurations are given in each model's subsection.

A single stratified 60/20/20 split of the cleaned CICIDS2017 dataset is created, and the indices persisted. For the CNN and DBN, these same indices are reused. Standardisation (StandardScaler) is fit on the training partition only and applied unchanged to validation and test to avoid leakage.

For the CNN-LSTM, a sequence model, non-overlapping windows of length $T = 10$ are built over the full, ordered standardised array and labelled by the last time step. A stratified 60/20/20 split at the window level is performed, which differs from the flat split used for the CNN and DBN. Training windows are lightly balanced by duplicating attack windows by a factor of 2, to mitigate the class imbalance effects, while validation and test windows preserve the natural skew.

Each model was trained using its setup described in Section 3. Early-stopping was used to monitor the designated validation criterion per model with best-weight restore.

Models are evaluated once on the untouched test-set. Accuracy, precision, recall, macro-F1, and per-class confusion matrices are computed. Confusion matrices are analysed to identify systematic failures such as frequently confused class as well as robust areas such as classes classified near-perfectly. These diagnostics can inform which architecture may be most suitable for optimisation and deployment, explored in Section 4.4.

To enable reproduction of inference pipelines, trained weights, the fitted scaler and label encoder, feature names, and the fixed split indices (flat and window-level) are persisted.

4.4 Post-Training Quantisation

This experiment evaluates how model size and performance change with varying post-training quantisation precisions, directly addressing Research Question 2. The experiment compares Float32 serving as the baseline, Float16 using weights, dynamic-range INT8 using INT8 weights and float activations, and finally INT8 FULL, where applicable, to identify a configuration that balances accuracy and resource usage. The fixed splits, scaler, and label mapping from Section 4.3 are reused.

TensorFlow models (CNN, CNN–LSTM): From the trained Keras models, four TFLite variants are exported: (i) Float32, (ii) Float-16 (weights), (iii) INT8 dynamic range (INT8 weights, and float activations), and (iv) INT8 Full. INT8 Full uses a 200 sample representative dataset for calibration and is skipped if an op lacks INT8 support. For CPU portability, the CNN-LSTM's LSTM layers are cloned with $unroll = True$ so that the graph runs with the XNNPACK CPU delegate. If a float graph requires non-core op, SELECT_TF_OPS are allowed, however INT8 graphs remain strictly INT8.

PyTorch model (DBN): The supervised head is reconstructed exactly as is done for training. This consisted of two sigmoid feature layers of 52 to 64 to 32, followed by ReLu, Dropout, Linear, ReLu, Dropout, and Linear layers. The best Float32 state dict was loaded, and three precisions were measured: (i) Float32, (ii) Float16 (weights), and (iii) INT8 Dynamic. PyTorch full per-tensor INT8 with static calibration is not used.

Inference and measurement were performed by small, deterministic runners that enforced CPU execution, disabling GPU usage, and capped threading to one. This was done by setting CUDA_VISIBLE_DEVICES=-1, TF device hiding, torch.set_num_threads(1), and enforcing the TFLite interpreter to use one thread with XNNPACK on CPU. Benchmarks run on a 2-vCPU Intel Xeon @ ~2.20 GHz VM with 12.7 GB RAM (see Appendix B.1 for exact environment). This environment was comparable to an entry-level dual-core system.

Each configuration was run three times and the medians were reported. The CNN measures latency batch 1 and throughput batch 256 (which has an automatic fallback ladder by a factor of 2, if an INT8 graph rejects large batches). The experiment measures a latency batch (b=1) and a larger throughput batch to probe capacity: CNN b=1, 256; CNN–LSTM b=1, 64; DBN b=1, 1024 (with an automatic fallback ladder if an INT8 graph rejects large batches). Latency (b=1) reflects edge devices that classify one flow at a time, whereas larger throughput batches test upper-bound processing capacity under batching. Although both are recorded for completeness, the results and discussion focus on b=1.

For each run, on-disk size (MB), wall time for a full test pass, process CPU time (user and system), throughput (samples/s), average latency (ms/sample), ΔRSS, and peak RSS are recorded. Predictions are saved and evaluated separately to compute accuracy, macro-F1, and per-class confusion matrices. All artifacts including TFLite files, JIT for DBN, reports, and JSON/CSV summaries are persisted for reproducibility.

4.5 Generalisation to Unseen Data

To probe zero-day robustness, with the aim of addressing of Research Question 3, in each run one attack family is treated as *unseen*. Two classes are evaluated separately, *DoS* and *Bots*. For a given held-out family c, all c-samples are excluded from training and validation only. The test-set is left unchanged so that it contains both seen attacks and the unseen family. The 2D CNN described in Section 3.3.1 is introduced in this experiment.

The CICIDS2017 preprocessing and splits are reused once again from Section 4.3 (single StandardScaler fit on train and a fixed LabelEncoder). The 1D CNN, 2D CNN and DBN consume flat per-flow features, while the CNN–LSTM uses the saved non-overlapping windows (T=10, stride 10) with a singleton channel. To control imbalance in training, the previously prepared ADASYN training arrays are kept for CNN/DBN, and for CNN–LSTM non-Normal windows are duplicated by a factor of 2. No resampling is applied to validation/test across all models.

All models remain as single-head multi-class Softmax classifiers. For novelty scoring, the *attack score* $s = 1 - p(\text{Normal})$ is used. A threshold τ is selected from Normal-score quantiles on the validation set to target a desired false-positive rate (FPR) on Normal. In

this case, 0.15 has been selected for DoS and 0.30 for Bots. The nearest threshold achieving the class-specific target is chosen. At the chosen τ^* Normal-vs-Unseen detection is reported and a 2×2 confusion matrix is computed. Closed-set performance on in-distribution classes (excluding the unseen rows) is also reported via accuracy and macro-F1. The full multi-class confusion matrix on the entire test-set is saved and the per-class distribution of predictions for the true unseen rows.

5 RESULTS AND DISCUSSION

5.1 Benchmarking Models

Following the setup outlined in 4.3 test-set results are reported for the Baseline CNN, CNN-LSTM and DBN to benchmark performances before applying quantisation methods or generalisation experiments.

Across the test-set, CNN–LSTM yields the best overall performance with 99.58% accuracy and very low FAR (0.27%), while maintaining balanced class behaviour. The Baseline CNN achieves high attack recall but incurs a higher FAR (6.34%) concentrated in minority and Normal confusions. The DBN attains 91.07% accuracy with the highest FAR (10.40%) and the largest per-class variance. Table 5.1 summarises, the key metrics achieved by all three models.

Model	Overall Accuracy	Weighted Precision	Weighted Recall	Weighted F1	False Alarm Rate	False Negative Rate
Baseline CNN	94.47%	97.00%	94.00%	96.00%	6.34	1.53%
CNN-LSTM	99.58%	≈100%	≈100%	≈100%	0.27%	1.1%
DBN	91.07%	96.00%	91.00%	93.00%	10.40%	0.85%

Table 5.1: Key Metrics across Models.

The Baseline CNN attains 94.47% accuracy but with pronounced per-class variability. The macro-F1 of 0.70 versus the weighted-F1 of 0.96 indicates uneven performance across classes due to label imbalance, despite ADASYN. The model is sensitive to attacks (attack detection rate ≈ 98.47%) but this comes with a high false-alarm rate, where 26 574 Normal traffic samples are flagged as attacks.

The model identifies major classes such as DDoS and DoS particularly well. In contrast, the minority classes (Bots and Web Attacks) are over-predicted shown by their high recalls of 0.99 and 0.97, but very low precisions of 0.03 and 0.16. Of the 26 574 Normal samples misclassified as attacks, 10 994 were Bots and 2 207 were Web Attacks. Predictions of the Normal class achieve precision of ≈ 1.00 but recall of 0.94, reflecting that some Normal flows are mislabelled as attacks. Overall, the baseline is a high-recall detector that catches majority of attacks, at the cost of inflated alerts for minority classes.

The CNN-LSTM achieves 99.58% accuracy with balanced behaviour across classes, seen with weighted precision, recall, and F1 ≈ 1.00. Compared to the Baseline CNN, this improvement is largely attributable to oversampling of attack windows after sequencing. During testing, the removal of this balancing recovered the baseline's minority-class precision pattern. The number of false positives is also much lower than the baseline, with only 114 Normal samples misclassified as attacks. The confusion matrix shows excellent detection of DDoS and DoS with minimal false positives/negatives and precision ≈ 1.00. These patterns suggest that temporal structure plus window balancing cleanly separates minority attacks without inflating false positives on the Normal class.

The DBN achieves 91.07% accuracy with less uniform class-wise behaviour. It performs strongly on DDoS (F1 = 0.91). The weighted-F1 of 0.93 versus macro-F1 of 0.62 indicates substantial per-class variability. The model detects minority classes like Bots and Web Attacks well (recalls ≈ 0.99), and maintains high recall for DDoS, DoS and Port Scanning (≈ 1.00, 0.98, 0.98).

However, false positives on Normal are substantial with 13 385 Normal flows are misclassified as Bots and 12 858 as DoS. False negatives are experienced less. Normal precision is ≈ 1.00, but its lower recall shows difficulty separating some Normal traffic from attacks.

Taken together, the CNN–LSTM establishes a very strong benchmark with minimal minority/Normal confusion, while the Baseline CNN and DBN would benefit from measures that diminish Normal-class false positives.

5.2 Quantisation of Models

The Baseline CNN, CNN-LSTM and DBN architectures are evaluated under Float32, Float16, and INT8 Dynamic (weight-only), and INT8 Full, where supported. Accuracy, macro-F1, latency (ms), and storage (MB) are measured. Overall, CNN–LSTM INT8 Dynamic offers the best edge trade-off.

Table 5.2 summarises closed-set quality (accuracy, macro-F1) and binary Normal/Attack operating behaviour (FAR/FNR). Performance is minimally affected under Float16 and INT8 Dynamic since accuracy (0.9959), FAR (0.27%), and FNR (1.1%) remain the same. DBN behaves similarly across Float32, Float16, and INT8 Dynamic (accuracy ≈0.911, macro-F1 ≈0.62, FAR ≈10.3–10.4%, FNR 0.85%), with a negligible accuracy difference for DBN INT8 Dynamic. For the CNN, INT8 Dynamic trades recall for more alarms. FNR improves from 1.53% to 0.48% but FAR rises from 6.34% to 6.92%. CNN INT8 Full quantisation fails, with a prominent drop in accuracy of 52.28% and an 81.28% decline in Macro-F1. Although the CNN-LSTM's INT8 Full does not fail, it does degrade accuracy by 0.84% and Macro-F1 by 7.2%, and raises FNR by 6.2 times. DBN INT8 Full is unsupported in this stack.

Figures 5.1 and 5.2 present Float32-normalised latency and storage where lower is better. Absolute memory (RSS, MB) and CPU% tables appear in the Appendix C.1 and C.2. Deployability is summarised here via normalized latency and storage. INT8 Dynamic performs the best in terms of size, such that the weights-only quantisation shrinks file size substantially. The CNN by ≈ 0.68×, CNN-LSTM by ≈ 0.49×, and DBN ≈ 0.39× relative to Float32. In the case of the CNN-LSTM and DBN, accuracy is unaffected. In terms of speed, the CNN-LSTM INT8 Dynamic performs the fastest. Latency drops to ≈ 0.77× Float32 (0.188 ms vs 0.244 ms), with Float16 also

Model	Precision	Accuracy	Macro-F1	FAR (%)	FNR (%)
CNN	Float32	0.9447	0.7003	6.34	1.53
	Float16	0.9447	0.7003	6.3	1.5
	INT8 Dynamic	0.9410	0.6935	6.92	0.48
	INT8 Full	0.4508	0.1311	49.8	53.7
CNN-LSTM	Float32	0.9959	0.9101	0.27	1.1
	Float16	0.9959	0.9102	0.27	1.1
	INT8 Dynamic	0.9959	0.9078	0.27	1.1
	INT8 Full	0.9875	0.8385	0.12	6.8
DBN	Float32	0.9107	0.6205	10.4	0.85
	Float16	0.9107	0.6206	10.4	0.85
	INT8 Dynamic	0.9117	0.6201	10.3	0.85

Table 5.2: Quantisation Results at Batch Size=1.

slightly faster. However, the DBN slows under quantisation. Float16 is $\approx 1.24\times$ slower and INT8 Dynamic is $\approx 2.90\times$ slower than Float32 likely due to overhead dominating tiny dense layers. The CNN's latency rises for INT8 Dynamic and INT8 Full which may be due to the dynamic dequantisation and quantisation overhead on the small convolutional network.

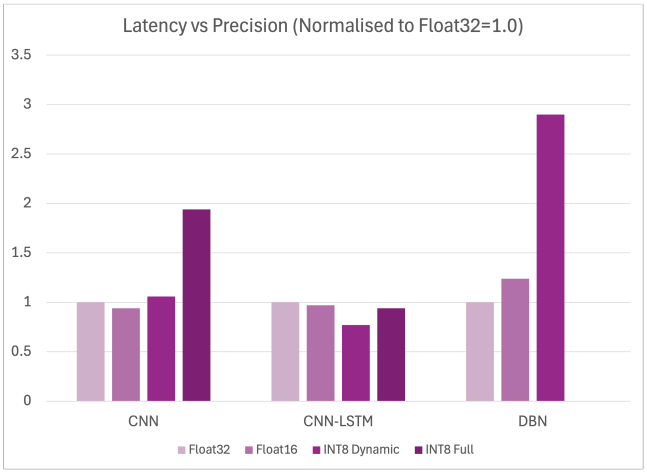

Figure 5.1: Latency vs Precision.

For the Python runtime used in this experiment, peak process RSS was dominated by the interpreter rather than model tensors, since it remained flat within each model across precisions. Hence, Float32 to Float16 or INT8 changes produce small ΔRSS for CNN and CNN-LSTM. DBN, however, is the exception. INT8 Dynamic reduces ΔRSS by $\approx 51\%$, suggesting DBN's working set ($weights + activations + buffers$) fits better into cache when weights are 8-bit. CPU time (ms/sample) was tracked, but closely followed latency across models. Per-core utilisation, measured per run, remained around the 98-99% thresholds, indicative that they are CPU-bound.

Full 8-bit quantisation requires quantised activations and operator coverage. Where integer kernels are missing, runtimes insert float fallbacks and per-tensor scale/dequant ops, and may be why INT8 Full underperforms in this experiment. For the CNN this causes both quality collapse and slower latency due to extra conversions.

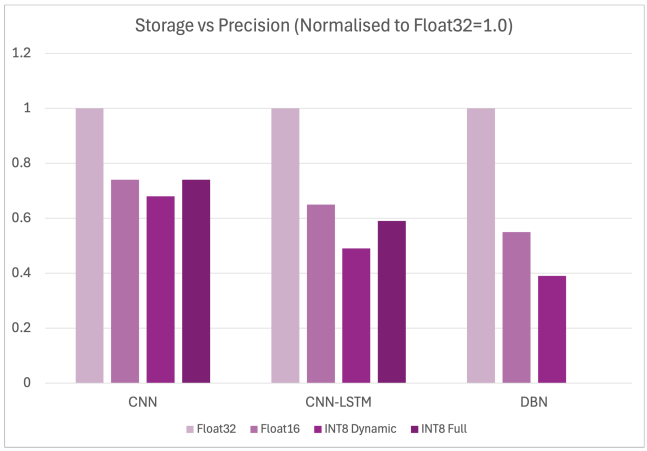

Figure 5.2: Storage vs Precision.

For CNN-LSTM it degrades macro-F1 and raises FNR, despite modest latency gains. Without Quantisation-Aware Training (QAT) to adapt activation ranges, Full-INT8 is risky. QAT works by allowing the model to adapt to reduced precision while training. This requires retraining and more computation, thus it was not explored in this study.

In terms of resource-constrained IDSs, these results suggest that INT8 Dynamic models strike a balance between performance, memory, and processing. Specifically, a CNN-LSTM INT8 Dynamic offers the best balance since it is smaller, faster, and almost achieves lossless accuracy. A Float16 CNN appears to be a safe compression for the baseline, reducing storage by 25%, with identical accuracy to its Float32 version but with a tiny speed-up. CNN INT8-Dynamic's higher FAR can be mitigated by threshold re-tuning or class-weighted thresholds without losing its recall benefit. INT8 Dynamic may be preferred if recall is of higher priority than speed. DBN is quantisation-capable in terms of storage, but slows runtime quite significantly. When single-sample latency is important, the Float32 is suggested based on these results.

5.3 Generalisation to Unseen Data

Generalisation was assessed by removing one attack class from training and exposed only at test time. A single operating threshold τ^* is chosen on validation to meet a target false-positive rate (FPR) on Normal traffic (Bots: 0.30 and DoS: 0.15). At test, samples are scored with $s = 1 - p(\text{Normal})$ and flagged as unknown when $s \geq \tau^*$.

Figure 5.3 displays the share of unseen samples flagged as malicious at the fixed FPR. Adding 2D convolutions materially improves open-set sensitivity. The 2D CNN has the smallest "Missed" segment for both hold-outs at the same FPR budgets, consistent with 2D convolutions capturing cross-feature correlations that separate novel behaviours from Normal. The 1D CNN and CNN–LSTM tend to be more sensitive to Bot-like deviations, whereas the DBN more readily treats broad traffic-level shifts (DoS) as non-Normal under

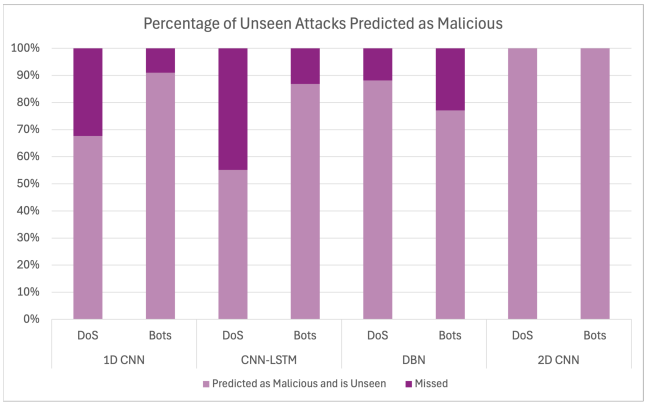

Figure 5.3: Percentage of Unseen Attack Class Predicted as Malicious.

a stricter FPR.

Held-out attack (FPR budget)	1D CNN	CNN-LSTM	DBN	2D CNN
Bots (FPR = 0.30)	0.91	0.87	0.77	1.0
DoS (FPR = 0.15)	0.68	0.55	0.88	1.0

Table 5.3: Unknown True Positive Rate at the Policy–Fixed False-Positive Rate.

The 1D CNN, with Bots held out and FPR=0.30, achieves a high unknown detection with TPR=0.91. In the multiclass view, many Bots are assigned to Normal by argmax, but their low Normal probability lifts s above τ^*, recovering them as unknown. With DoS held out at FPR=0.15, TPR drops to 0.68 and unseen DoS partially collapses into Normal with some spill to DDoS. This indicates difficulty separating traffic-level bursts from benign flows under a tighter alert budget.

The CNN-LSTM preserves strong closed-set structure, but zero-day sensitivity remains FPR-dependent. A TPR of 0.87 is achieved on Bots at FPR=0.30 and 0.55 on DoS at FPR=0.15. As with the 1D CNN, argmax often labels Bots as Normal, yet scoring by s recovers a portion at the operating threshold. The model remains strong on the seen classes, reflecting the benefit of temporal modelling for seen traffic structure. Overall, the CNN-LSTM excels on seen traffic, but its zero-day sensitivity depends heavily on the FPR.

The DBN is comparatively better on unseen DoS achieving a TPR of 0.88 at FPR=0.15 and weaker on Bots with 0.77 at FPR=0.30. This aligns with its decision boundaries that more readily flag broad traffic shifts as non-Normal, which is useful in detecting DoS. Its higher false-alarm tendency on Normal constrains usable FPR in practice. On the seen classes, closed set accuracy is 0.9146 with macro-F1 at 0.639 with persistent false positives on Normal.

The 2D CNN variant delivers the highest unknown-attack TPR on both hold-outs (1.00 at the same FPR budgets), matching the qualitative drop in "Missed" in Figure 5.3. This suggests that cross-feature receptive fields materially help distinguish novel behaviours from Normal at fixed alert budgets.

Overall, no single architecture necessarily dominates at all operating points. However, performance depends on both the attack type and the allowable FPR. The 2D CNN is the most sensitive at the chosen false-positive rates, the DBN is strongest on unseen DoS under stricter FPR, and the CNN–LSTM offers the best preservation of closed-set structure. In deployment, architecture choice should be paired with explicit threshold calibration to the target false-positive budget. The results also suggests that a 2D variant of the 1D CNN merits inclusion in all experiments, to rival the CNN-LSTM's balanced performance, although it was not explored in this study.

6 CONCLUSION

This study investigated whether a lightweight AI-IDS can meet edge constraints while retaining zero-day sensitivity. Across benchmarking, quantisation, and generalisation experiments, the results show that it is feasible, but architecture and operating points thresholds matter.

Among the evaluated models, the CNN-LSTM offers the best overall resource-accuracy trade-off. It retains near loss-less closed-set accuracy, and when quantised, produces a compact and fast model suitable for constrained CPUs. The CNN remains a robust baseline, but it is less stable under aggressive quantisation. DBN quantisation produces small model files, but slows down runtime and heightens false-alarm rate on Normal Traffic.

Quantisation results suggest that Float16 is a safe default across models, giving storage reductions without measurable accuracy loss across models. INT8 Dynamic provides the largest size reductions and, for CNN–LSTM, material latency gains while maintaining detection quality. In contrast, Full-INT8 degraded accuracy and increased false negatives, thus should be avoided unless Quantisation-Aware Training is used to recover performance.

Generalisation, an important quality in intrusion detection, is strongly dependent on the false-positive rate (FPR). At fixed FPRs, models emphasise different unknown attacks. The DBN is more responsive to traffic-level shifts (DoS) under stricter FPRs, whereas the 1D CNN and CNN-LSTM respond to Bot-like deviations at looser budgets. Importantly, introducing a 2D variant of the CNN substantially improved open-set sensitivity, achieving the highest unknown-attack TPR at the same FPR budgets—evidence that cross-feature receptive fields help separate novel behaviours from Normal. While the 2D CNN was not fully benchmarked or quantised, these results suggest it merits inclusion alongside CNN–LSTM in edge-focused designs. In practice, operating thresholds should be calibrated on validation to the deployment's false-positive rate, and potentially re-tuned after quantisation.

Overall, lightweight AI-IDS is feasible. The study finds a quantised CNN-LSTM, specifically INT8 Dynamic, performs best with false-positive rate tuning for generalisation, and suggests the exploration of 2D CNN variants to further boost sensitivity to unknown attacks.

6.1 Limitations and Future Work

Memory was measured via ΔRSS on a Python host, which is dominated by interpreter overhead, thus on-device RAM savings from quantisation are therefore likely under-reported. Experiments ran on a single CPU and since latency and kernel coverage vary by processor, results are not hard-ware independent. $Batchsize = 1$ is edge-realistic but can penalise architectures with high per-inference overheads. Evaluation used only CICIDS2017 with two held-out families (Bots, DoS), leaving other attack types, traffic mixes, and measurement noise untested. Methodologically, the study focused on post-training quantisation. Recurrent layers and small Multi Layer Perceptrons can be sensitive to post-training quantisation, and may have benefitted from Quantisation-Aware Training. ADASYN improved class balance but can reshape decision boundaries.

Future work can therefore measure static and peak memory with a fixed arena and profile latency and energy on representative Microcontrollers, ideally in virtual or live network deployments to validate resource budgets. Broader evaluation across additional datasets, temporally separated splits, and more leave-one-attack-out scenarios would strengthen claims of generalisation. On the modelling side, applying Quantisation Aware Training and mixed-precision schemes could recover full-INT8 accuracy and reduce activation RAM. Finally, future work can combine quantisation with pruning or structured sparsity and knowledge distillation may further compress models without sacrificing recall.

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] Abdullah Aljumah. 2021. IoT-based intrusion detection system using convolution neural networks. *PeerJ Computer Science* 7 (2021), e721.

[3] Zainab Alwaisi, Tanesh Kumar, Erkki Harjula, and Simone Soderi. 2024. Securing constrained IoT systems: A lightweight machine learning approach for anomaly detection and prevention. *Internet of Things* 28 (2024), 101398.

[4] Eric Anacleto. 2025. CICIDS2017: Cleaned & Preprocessed. https://www.kaggle.com/datasets/ericanacletoribeiro/cicids2017-cleaned-and-preprocessed. Kaggle dataset; preprocessed release derived from CICIDS2017. Accessed: 2025-07-25.

[5] Jahongir Azimjonov and Taehong Kim. 2024. Designing accurate lightweight intrusion detection systems for IoT networks using fine-tuned linear SVM and feature selectors. *Computers & Security* 137 (2024), 103598.

[6] Shahid Allah Bakhsh, Muhammad Almas Khan, Fawad Ahmed, Mohammed S Alshehri, Hisham Ali, and Jawad Ahmad. 2023. Enhancing IoT Network security through deep learning-powered Intrusion Detection System. *Internet of Things* 24 (2023), 100936.

[7] Nagaraj Balakrishnan, Arunkumar Rajendran, Danilo Pelusi, and Vijayakumar Ponnusamy. 2021. Deep Belief Network enhanced intrusion detection system to prevent security breach in the Internet of Things. *Internet of things* 14 (2021), 100112.

[8] Othmane Belarbi, Aftab Khan, Pietro Carnelli, and Theodoros Spyridopoulos. 2022. An intrusion detection system based on deep belief networks. In *International conference on science of cyber security*. Springer, 377–392.

[9] 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

[10] Md Moin Uddin Chowdhury, Frederick Hammond, Glenn Konowicz, Chunsheng Xin, Hongyi Wu, and Jiang Li. 2017. A few-shot deep learning approach for improved intrusion detection. In *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*. IEEE, 456–462.

[11] Jiawei Du, Kai Yang, Yanjing Hu, and Lingjie Jiang. 2023. NIDS-CNNLSTM: Network intrusion detection classification model based on deep learning. *IEEE Access* 11 (2023), 24808–24821.

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

[13] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2016. An evaluation framework for intrusion detection dataset. In *2016 International conference on information science and security (ICISS)*. IEEE, 1–6.

[14] Mohammed Sayeeduddin Habeeb and T Ranga Babu. 2022. Network intrusion detection system: a survey on artificial intelligence-based techniques. *Expert Systems* 39, 9 (2022), e13066.

[15] Asmaa Halbouni, Teddy Surya Gunawan, Mohamed Hadi Habaebi, Murad Halbouni, Mira Kartiwi, and Robiah Ahmad. 2022. CNN-LSTM: hybrid deep neural network for network intrusion detection system. *IEEE Access* 10 (2022), 99837–99849.

[16] 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.

[17] Belal Ibrahim Hairab, Heba K Aslan, Mahmoud Said Elsayed, Anca D Jurcut, and Marianne A Azer. 2023. Anomaly detection of zero-day attacks based on CNN and regularization techniques. *Electronics* 12, 3 (2023), 573.

[18] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. pmlr, 448–456.

[19] Mohammed Jouhari and Mohsen Guizani. 2024. Lightweight cnn-bilstm based intrusion detection systems for resource-constrained iot devices. In *2024 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 1558–1563.

[20] 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)*. IEEE, 1–6.

[21] Naila Marir, Huiqiang Wang, Guangsheng Feng, Bingyang Li, and Meijuan Jia. 2018. Distributed abnormal behavior detection approach based on deep belief network and ensemble SVM using spark. *IEEE Access* 6 (2018), 59657–59671.

[22] Michal Markevych and Maurice Dawson. 2023. A review of enhancing intrusion detection systems for cybersecurity using artificial intelligence (ai). In *International conference knowledge-based organization*, Vol. 29. 30–37.

[23] Nour Moustafa and Jill Slay. 2016. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective* 25, 1-3 (2016), 18–31.

[24] X.-H. Nguyen, X.-D. Nguyen, H.-H. Huynh, and K.-H. Le. 2022. Realguard: a Lightweight Network Intrusion Detection System for IoT Gateways. *Sensors* 22, 2 (2022), 432. https://doi.org/10.3390/s22020432

[25] Debanjana Niogi, Dr Devender Kumar, Deep Ranjan, and Jasveer Singh. 2023. Recent Advances and Future Directions in Ai-Based Intrusion Detection Systems for Network Security. *Available at SSRN 4485452* (2023).

[26] Mathappan Nivaashini, E Suganya, S Sountharrajan, M Prabu, and Durga Prasad Bavirisetti. 2024. FEDDBN-IDS: federated deep belief network-based wireless network intrusion detection system. *EURASIP Journal on Information Security* 2024, 1 (2024), 8.

[27] 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.

[28] AM Oyelakin, AO Ameen, TS Ogundele, T Salau-Ibrahim, UT Abdulrauf, HI Olufadi, IK Ajiboye, S Muhammad-Thani, et al. 2023. Overview and exploratory analyses of CICIDS2017 intrusion detection dataset. *Journal of Systems Engineering and Information Technology (JOSEIT)* 2, 2 (2023), 45–52.

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

[30] Emad Ul Haq Qazi, Abdulrazaq Almorjan, and Tanveer Zia. 2022. A one-dimensional convolutional neural network (1D-CNN) based deep learning system for network intrusion detection. *Applied Sciences* 12, 16 (2022), 7986.

[31] Md Mahbubur Rahman, Shaharia Al Shakil, and Mizanur Rahman Mustakim. 2025. A survey on intrusion detection system in IoT networks. *Cyber Security and Applications* 3 (2025), 100082.

[32] B Ravinder Reddy. 2024. Network Intrusion Detection using Machine Learning. (2024).

[33] Vignesh Reddy, R Sunitha, 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)*. IEEE, 1–6.

[34] Souradip Roy, Juan Li, Bong-Jin Choi, and Yan Bai. 2022. A lightweight supervised intrusion detection mechanism for IoT networks. *Future Generation Computer Systems* 127 (2022), 276–285.

[35] Kshira Sagar Sahoo, Bata Krishna Tripathy, Kshirasagar Naik, Somula Ramasubbareddy, Balamurugan Balusamy, Manju Khari, and Daniel Burgos. 2020. An evolutionary SVM model for DDOS attack detection in software defined networks. *IEEE access* 8 (2020), 132502–132513.

[36] 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.

[37] T. Sowmya and E.A. Mary Anita. 2023. A Comprehensive Review of AI Based Intrusion Detection System. *ScienceDirect* 28 (2023), 100827–100827. https://doi.org/10.1016/j.measen.2023.100827

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

[39] Pengfei Sun, Pengju Liu, Qi Li, Chenxi Liu, Xiangling Lu, Ruochen Hao, and Jinpeng Chen. 2020. DL-IDS: Extracting Features Using CNN-LSTM Hybrid Network for Intrusion Detection System. *Security and communication networks* 2020, 1 (2020), 8890306.

[40] Xiaopeng Tan, Shaojing Su, Zhen Zuo, Xiaojun Guo, and Xiaoyong Sun. 2019. Intrusion detection of UAVs based on the deep belief network optimized by PSO. *Sensors* 19, 24 (2019), 5529.

[41] 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.

[42] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. 2019. Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology* 17, 1 (2019), 26–40.

[43] Yihan Xiao, Cheng Xing, Taining Zhang, and Zhongkai Zhao. 2019. An intrusion detection model based on feature reduction and convolutional neural networks. *IEEE Access* 7 (2019), 42210–42219.

[44] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. 2017. A deep learning approach for intrusion detection using recurrent neural networks. *Ieee Access* 5 (2017), 21954–21961.

[45] Guangzhen Zhao, Cuixiao Zhang, and Lijuan Zheng. 2017. Intrusion detection using deep belief network and probabilistic neural network. In *2017 IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC)*, Vol. 1. IEEE, 639–642.

[46] Ruijie Zhao, Guan Gui, Zhi Xue, Jie Yin, Tomoaki Ohtsuki, Bamidele Adebisi, and Haris Gacanin. 2021. A novel intrusion detection method based on lightweight neural network for internet of things. *IEEE Internet of Things Journal* 9, 12 (2021), 9960–9972.

APPENDIX

A CONFUSION MATRICES

Below are the confusion matrices which further detail benchmarked model performance.

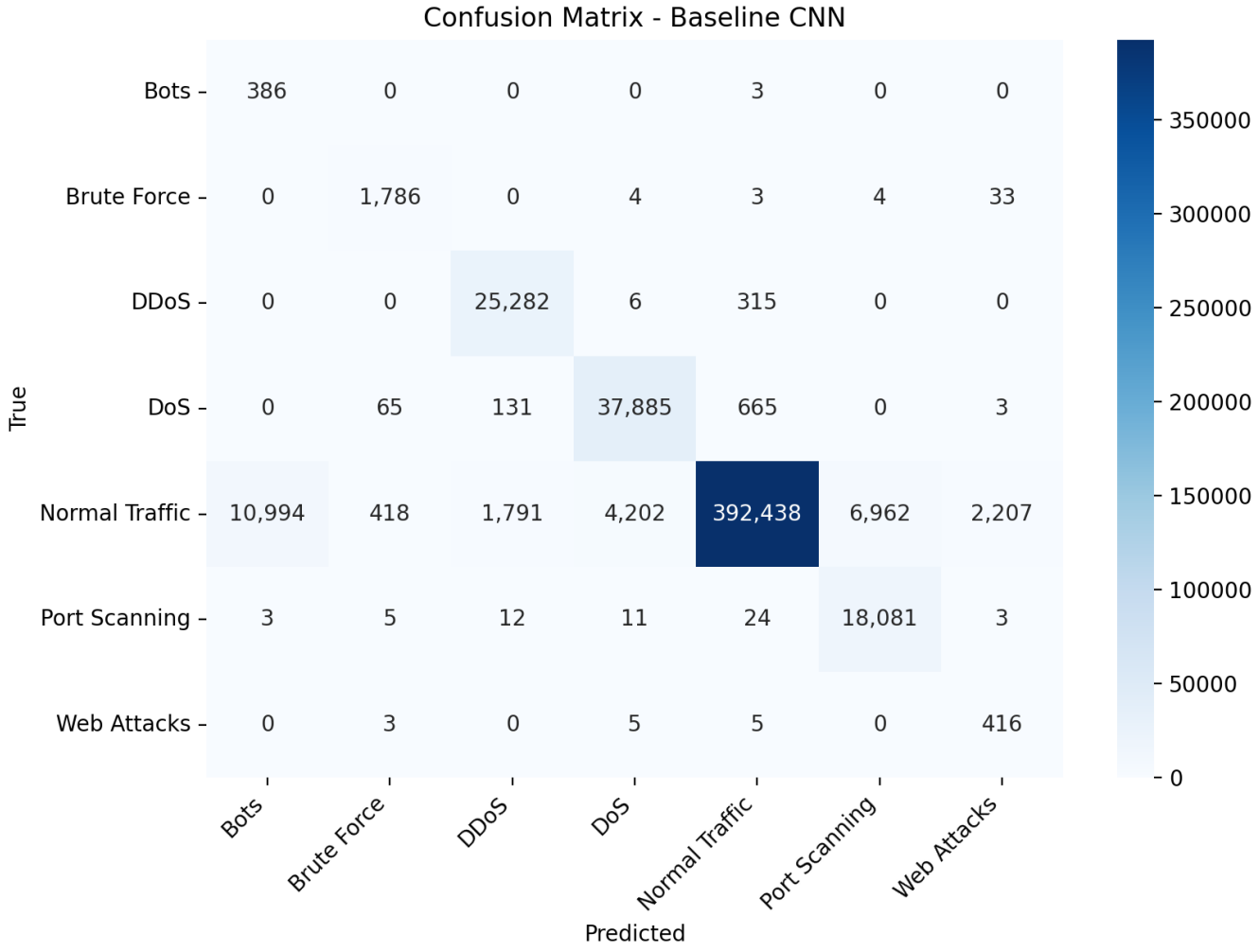

Figure A.1: Baseline CNN Confusion Matrix.

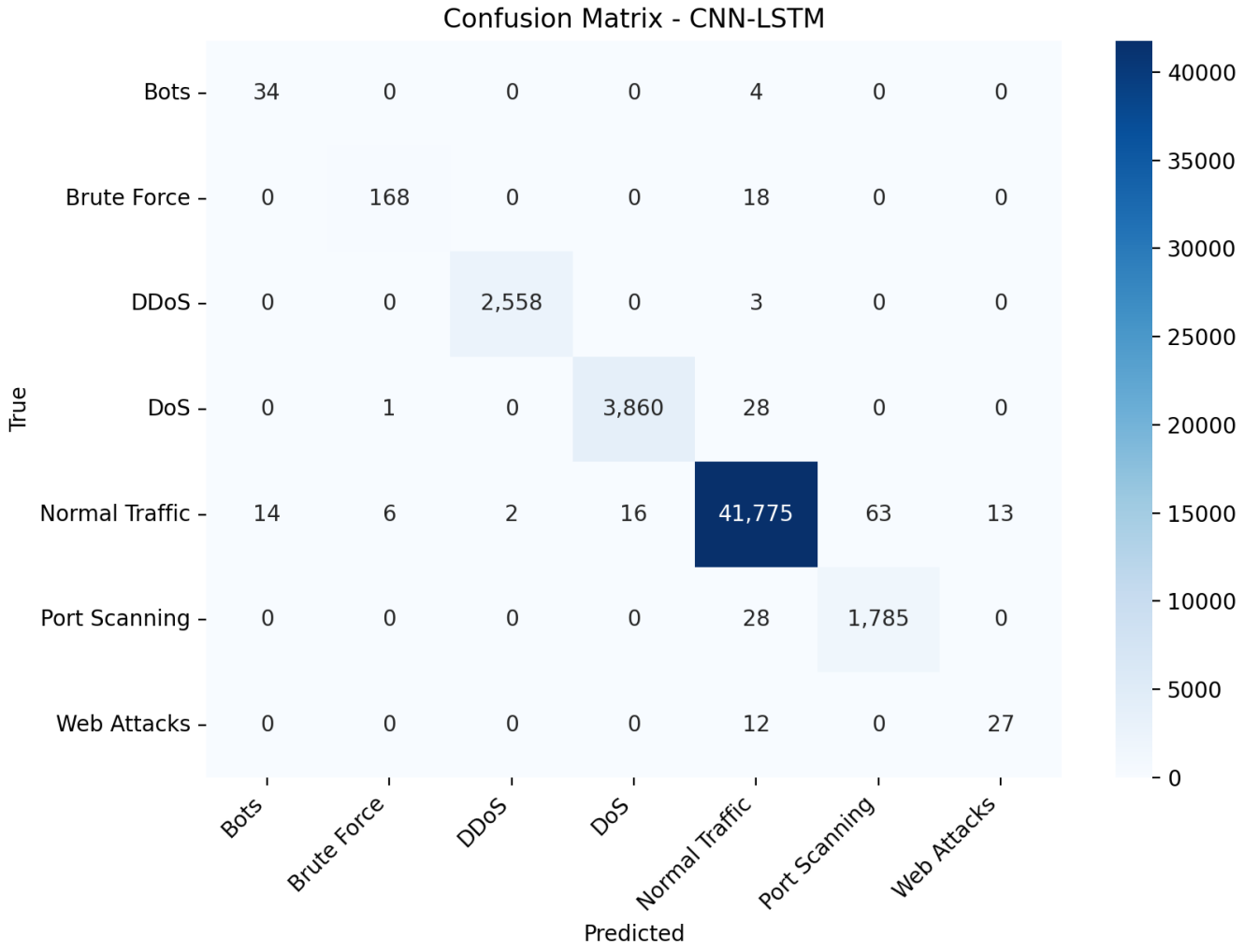

Figure A.2: CNN-LSTM Confusion Matrix.

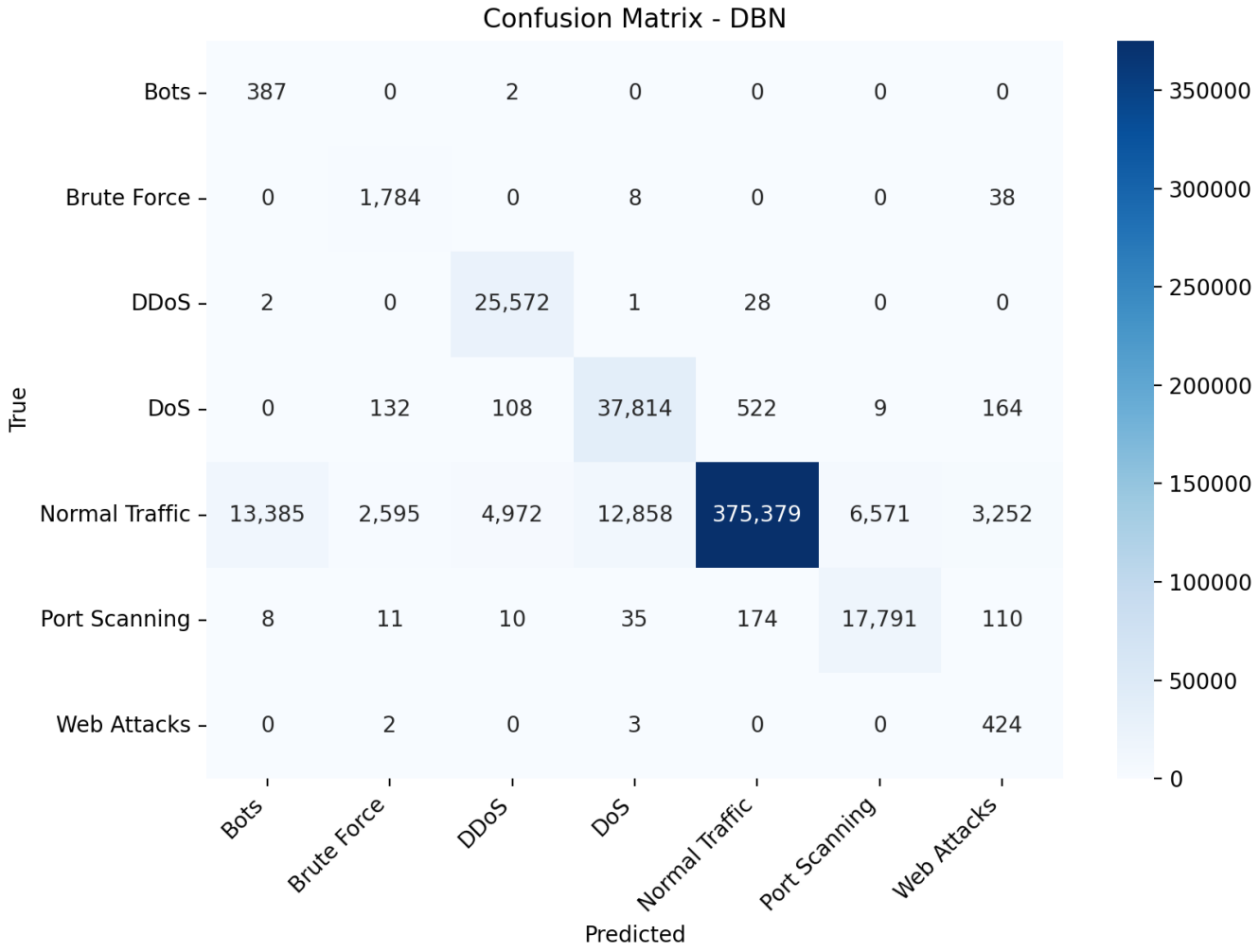

Figure A.3: DBN Confusion Matrix.

B HARDWARE AND SOFTWARE ENVIRONMENT

Below is the hardware and software environments under which the post-training quantisation was performed.

Component	Specification
Host type	KVM virtual machine (Linux 6.1)
CPU	Intel Xeon @ 2.20 GHz, 2 vCPU (AVX2/FMA)
Memory	12.7 GB RAM, no swap
Python	3.12.11
TensorFlow/TFLite	2.19.0 (XNNPACK CPU delegate)
PyTorch	2.8.0+cu126 (CPU backend)
NumPy / psutil	2.0.2 / 5.9.5
Threading policy	CPU-only; threads capped per experiment
Metrics collection	`psutil` (CPU time, RSS), POSIX `resource` (peak RSS)

Table B.1: Hardware and Software Environment.

C POST-TRAINING QUANTISATION RESULTS

Below is more detailed results of post-training quantisation in terms of memory footprint and processing usage.

Model	Precision	Storage (MB)	ΔRSS (MB)	ΔRSS vs F32 (%)
CNN	Float32	0.019	23.930	0.000
	Float16	0.014	23.742	0.786
	INT8 Dynamic	0.013	23.863	0.280
	INT8 Full	0.014	8.566	64.204
CNN-LSTM	Float32	0.278	1.879	0.000
	Float16	0.182	1.934	-2.927
	INT8 Dynamic	0.137	1.992	-6.014
	INT8 Full	0.163	2.000	-6.440
DBN	Float32	0.044	88.715	0.000
	Float16	0.024	95.988	-8.198
	INT8 Dynamic	0.017	43.176	51.332

Table C.1: Memory Footprint at Batch Size = 1.

Model	Precision	Latency (ms)	Δ Lat. vs F32 (%)	Throughput (sps)	Δ Thr. vs F32 (%)	CPU time (ms / sample)
CNN	Float32	0.017	+0.00	60374.71	+0.00	0.0164
	Float16	0.016	+5.88	60679.14	+0.50	0.0163
	INT8 Dynamic	0.018	-5.88	56096.21	-7.09	0.0176
	INT8 Full	0.033	-94.12	30553.60	-49.39	0.0322
CNN-LSTM	Float32	0.244	+0.00	4097.02	+0.00	0.2398
	Float16	0.236	+3.28	4234.36	+3.35	0.2319
	INT8 Dynamic	0.188	+22.95	5328.75	+30.06	0.1853
	INT8 Full	0.229	+6.15	4359.54	+6.41	0.2245
DBN	Float32	0.091	+0.00	10961.73	+0.00	0.0905
	Float16	0.113	-24.18	8874.62	-19.04	0.1092
	INT8 Dynamic	0.264	-190.11	3790.75	-65.42	0.2624

Table C.2: Processing Usage at batch size = 1.

D GENERALISATION RESULTS

Below is a more detailed split of how unseen attacks are predicted as malicious.

Model	Unseen	Predicted as Malicious & Unseen	Total Unseen	Missed	Percentage (%)
1D CNN	DoS	26,229	38,749	12,520	67.69
	Bots	354	389	35	91.00
CNN-LSTM	DoS	2,144	3,889	1,745	55.13
	Bots	33	38	5	86.84
DBN	DoS	34,165	38,749	4,584	88.17
	Bots	300	389	89	77.12
2D CNN	DoS	38,749	38,749	0	100.00
	Bots	389	389	0	100.00

Table D.1: Split of Total Unseen Attacks into Detected and Missed.