CMCSC22 BTP Project-I Report

Distilled Intelligence: Lightweight Intrusion Detection

Submitted in partial fulfillment of
the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

Submitted by
Group - 20

Roll No	Names of Students
2022UCM2305	Prabhmeet Singh
2022UCM2330	Aryan Jain
2022UCM2365	Radhacharan

Under the guidance of
Dr.Anand Gupta

Department of Computer Science and Engineering
Netaji Subhas University of Technology
Azad Hind Fauj Marg, Sec-3, Dwarka, New Delhi 110078

Odd Semester 2025-26

Department of Computer Science and Engineering

Netaji Subhas University of Technology

Certificate

This is to certify that this is a bonafide record of the project presented by the students whose names are given below during Odd Semester 2025-26 in partial fulfilment of the requirements of the degree of Bachelor of Technology in Computer Science and Engineering.

Roll No	**Names of Students**
2022UCM2305	Prabhmeet Singh
2022UCM2330	Aryan Jain
2022UCM2365	Radhacharan

Dr. Anand Gupta
(Project Supervisor)

Date: 8th December 2025

Contents

ABSTRACT

The continuous escalation in the volume of high-speed network traffic and the growing sophistication of contemporary cyber threats require the deployment of robust Network Intrusion Detection Systems (NIDS) as a foundational element of modern cybersecurity infrastructure. While Deep Learning (DL) models offer a demonstrably superior capability for the identification of complex, novel attack patterns, their practical deployment on resource-constrained platforms—such as edge computing nodes and Internet of Things (IoT) gateways—is significantly hampered by their substantial computational footprint and resultant high inference latency. To effectively address this critical trade-off, this project proposes a high-efficiency NIDS framework utilizing the established technique of Knowledge Distillation (KD) [12]. Using the comprehensive UNSW-NB15 dataset [1], we adopted a complex 1D-Convolutional Neural Network (1D-CNN) architecture based on the work of Vibhute et al. [3] as the sophisticated "Teacher" model, meticulously optimized for maximum detection fidelity via techniques including Batch Normalization [10]. Subsequently, the rich, nuanced predictive knowledge embedded within this teacher model was systematically transferred into a series of significantly more compact "Student" architectures through the minimization of the Kullback-Leibler (KL) divergence [13]. Experimental validation identifies two standout architectures tailored to specific deployment constraints. The R1_RobustTiny model achieved maximum storage efficiency, reducing the parameter footprint from 0.78 MB to 0.05 MB ($\approx$ 94% reduction). However, for dynamic operational environments, the R2_RobustSmall architecture emerged as the optimal solution. In a discrete-event Live Traffic Simulator, the R2 model demonstrated exceptional stability, achieving 100% detection accuracy with an average inference latency of 87.59 ms, successfully outperforming the baseline Teacher model in processing speed while retaining full classification fidelity. These findings strongly position Knowledge Distillation as an essential enabling technology for realizing high-performance, real-time intrusion detection systems within computationally resource-scarce environments [14], [16].

I. INTRODUCTION

Network Intrusion Detection Systems (NIDS) are a vital component of modern network security, continuously scrutinizing traffic to isolate suspicious activities [5]. Due to the inability of traditional signature-based systems to identify zero-day attacks, there has been a necessary shift toward anomaly detection using advanced Deep Learning (DL) algorithms [2]. This project's foundation lies in recent work that demonstrated the high detection efficacy of Convolutional Neural Networks (CNNs) for anomaly detection on the UNSW-NB15 dataset [3]. This accurate CNN architecture serves as the complex Teacher model for our research.

However, state-of-the-art DL models, including the CNN Teacher evaluated in [3], achieve high accuracy at the cost of architectural complexity. This complexity imposes a significant computational burden [11], leading to detrimental inference latency [18]. Such delays are critical in high-throughput real-time systems, risking packet drops and providing attackers with a window to breach the system before automated countermeasures activate [17]. This performance bottleneck represents the critical limitation that our project aims to resolve while preserving the high detection accuracy demonstrated by the foundational research.

To resolve the fundamental trade-off between high accuracy and operational efficiency on edge computing devices, this project thoroughly explores Knowledge Distillation (KD) [12]. The KD core mechanism trains a streamlined "Student" model to emulate the larger "Teacher" model's predictive behavior by transferring complex, learned information (or "dark knowledge") via soft probability outputs [12], [13]. This proposed framework facilitates the deployment of resilient [16], high-fidelity NIDS on hardware with extremely limited processing power, thereby ensuring robust network security without compromising essential network performance metrics [14].

II. MOTIVATION

The central motivation for this research stems from a critical and persistent challenge at the intersection of modern network security and operational efficiency: the difficulty of deploying high-fidelity, Deep Learning (DL)-based Network Intrusion Detection Systems (NIDS) within resource-constrained, real-time environments.

A. The Dilemma of High-Fidelity NIDS

The necessity for DL models in NIDS is established by their superior capability to perform anomaly detection, effectively identifying sophisticated zero-day attacks that bypass traditional signature-based methods [2], [5]. However, this enhanced detection fidelity is intrinsically linked to the model's structural complexity [11]. This high complexity translates into a substantial computational footprint and, critically, high inference latency, which is intolerable for real-time packet processing in high-throughput networks [18]. In distributed architectures, particularly those involving edge computing and IoT gateways, this computational burden renders large models practically unviable, creating a critical gap between theoretical performance and practical deployment feasibility [14].

B. Bridging the Accuracy-Efficiency Gap via KD

This project is strategically motivated by the need to resolve this fundamental accuracy-efficiency trade-off. We propose Knowledge Distillation (KD) [12] as the enabling technology to systematically compress the superior knowledge embedded within a complex "Teacher" 1D-CNN model [3] into a highly efficient "Student" architecture. The objective is to produce a lightweight model that aims to retain the teacher's high detection accuracy on the UNSW-NB15 dataset [1], while concurrently achieving radical compression in terms of storage and computational overhead. By targeting a significant reduction in model size and a corresponding increase in inference throughput, this research seeks to substantiate KD as a vital methodology for achieving ubiquitous, high-performance intrusion detection at the network edge [16].

III. LITERATURE SURVEY

The trajectory of Network Intrusion Detection System (NIDS) development has fundamentally shifted from reactive, signature-based methods to proactive, anomaly-based detection, necessitated by the increasing sophistication of cyber threats and the proliferation of zero-day attacks [5]. The adoption of Deep Learning (DL) architectures has yielded substantial improvements in the ability to detect these unknown anomalies by automatically extracting features from large network datasets [2]. Specifically, Convolutional Neural Networks (CNNs) have proven highly effective in this domain; Vibhute et al. [3] demonstrated the superior performance of 1D-CNNs on modern network traffic data, establishing a high-fidelity benchmark that serves as the foundational "Teacher" model for this research. Further supporting the efficacy of 1D-CNNs in intrusion detection, Qazi et al. [8] highlighted their ability to capture temporal correlations in packet flows.

Evaluating these advanced models requires robust datasets that reflect modern traffic characteristics. The UNSW-NB15 dataset [1] has emerged as a crucial modern benchmark. As analyzed by Moustafa and Slay [4], this dataset addresses the statistical shortcomings of the older KDD99 benchmark, providing a more realistic and complex basis for training high-performance NIDS models.

However, the high detection fidelity afforded by state-of-the-art DL models comes at the cost of significant structural complexity and a large parameter count [11]. This complexity results in an unacceptably high computational footprint and prolonged inference latency, issues that are fundamentally incompatible with the memory and processing constraints of decentralized edge computing and IoT gateway devices [14]. These latency constraints are particularly detrimental to high-throughput, real-time network monitoring, where delays can compromise the efficacy of the entire security posture [18], [17].

To resolve this critical efficiency bottleneck, the literature points towards model compression techniques. Among these, Knowledge Distillation (KD) stands out as the most promising paradigm for maintaining performance while reducing resource consumption [15]. Pioneered by Hinton et al. [12], KD involves the transfer of "dark knowledge" from a large, high-capacity "Teacher" network to a lightweight "Student" network. This transfer is mathematically governed by minimizing the Kullback-Leibler (KL) divergence [13] between the soft output distributions of the two models. Recent studies have successfully validated the application of KD for creating resilient, low-latency anomaly detection systems in IoT contexts [14], [16]. This survey establishes the necessity of applying KD to bridge the recognized accuracy-efficiency gap in modern NIDS deployment.

IV. PROBLEM STATEMENT

The increasing prevalence of high-volume, high-velocity network traffic, coupled with the sophisticated nature of modern cyber threats, mandates the use of highly accurate Deep Learning (DL)-based Network Intrusion Detection Systems (NIDS). However, while high-capacity models, such as the optimized 1D-Convolutional Neural Network (1D-CNN) proposed by Vibhute et al. [3], achieve state-of-the-art detection performance on complex datasets like UNSW-NB15 [1], this comes at the cost of a formidable computational footprint [11].

The central problem addressed by this project is the critical incompatibility between the high resource demands of high-fidelity DL-based NIDS and the severe operational constraints of resource-limited deployment environments, specifically edge computing and IoT gateways. This incompatibility manifests as an unacceptable increase in inference latency and a significant reduction in processing throughput [18], which severely compromises the ability of the NIDS to perform real-time packet inspection.

The research gap is defined as the lack of a proven, highly efficient model compression technique that can effectively transfer the complex, learned predictive knowledge of a large NIDS teacher model into a compact student architecture while maintaining near-lossless detection accuracy. Therefore, the goal is to validate Knowledge Distillation (KD) [12] as a systematic solution to produce a lightweight NIDS model capable of high-performance operation on constrained hardware, thereby resolving the fundamental trade-off between security efficacy and operational efficiency.

V. OBJECTIVES

The overarching objective of this BTP thesis is to develop and validate a highly efficient, high-performance Network Intrusion Detection System (NIDS) suitable for resource-constrained edge environments using the Knowledge Distillation (KD) paradigm. This primary goal is achieved through the following specific research and engineering objectives:

- **Knowledge Baseline Establishment:** Train and validate a robust 1D-CNN Teacher Model to achieve an F1-Score (> 0.95) on the *UNSW-NB15* dataset, establishing a high-fidelity source for knowledge transfer.
- **Input Space Optimization:** Systematically reduce input feature dimensionality using Random Forest Gini Impurity reduction to enhance computational efficiency for the Student Architectures.
- **Distillation Framework Tuning:** Implement and optimize the Knowledge Distillation (KD) regimen by integrating the Composite Loss Function and experimentally tuning hyperparameters (T and α) to maximize stable knowledge transfer.
- **Trade-off Analysis and Model Selection:** Design and benchmark compressed Student Architectures to quantify the Model Size vs. Detection Efficiency Trade-off, selecting the minimal-capacity model (*Student R2_RobustSmall*) that meets latency and accuracy goals.
- **Operational Validation and Metrics:** Empirically validate the optimal model under simulated edge deployment conditions by quantifying critical metrics: high-volume processing throughput and continuous inference latency (τ_{avg}).
- **Deployment Pipeline Engineering:** Construct a self-contained, reproducible inference pipeline by serializing all trained weights and preprocessing artifacts, ensuring immediate deployment and scalability.

VI. PROPOSED METHODOLOGY

This research develops a lightweight NIDS through Knowledge Distillation, optimizing the deployment of a high-capacity Teacher model by transferring its knowledge to a compact Student architecture.

A. Dataset Preparation and Exploration

- **Data Acquisition:** The UNSW-NB15 dataset was selected as the benchmark, with all four partitioned CSV files logically concatenated into a single dataframe, totaling approximately 2.54 million records denoted as $D = \{x_i, y_i\}_{i=1}^{N}$..
- **Target Definition:** Configured as a binary classification problem using the *label* attribute (0: Normal, 1: Attack).
- **Exploratory Data Analysis (EDA):** Conducted statistical assessment of class imbalance (Fig. 1), attack taxonomy (Fig. 2), feature correlation (Fig. 3), and density estimation (Fig. 4) for redundancy and distributional insight.

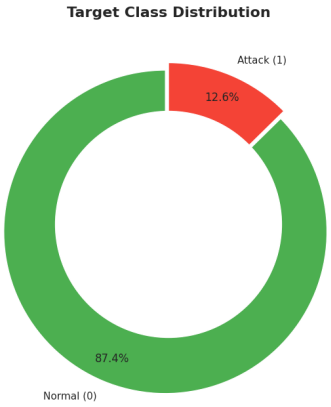

Figure 1: Binary Class Distribution of the UNSW-NB15, showing severe class imbalance (C_0 vs. C_1).

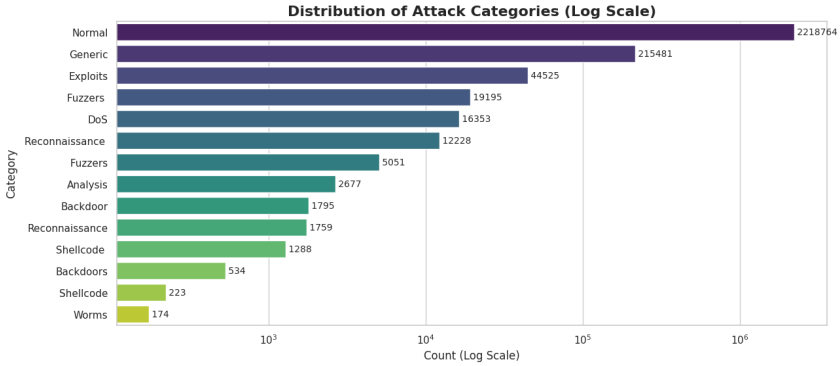

Figure 2: Attack Taxonomy Distribution (Log Scale), showing the frequency and relative scarcity of individual attack categories.

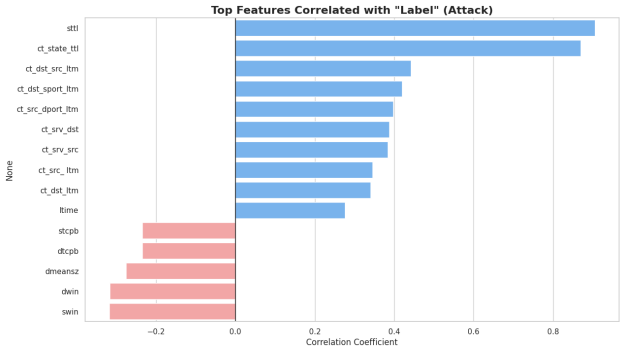

Figure 3: Pearson Correlation Heatmap, used to identify feature relationships and redundancy.

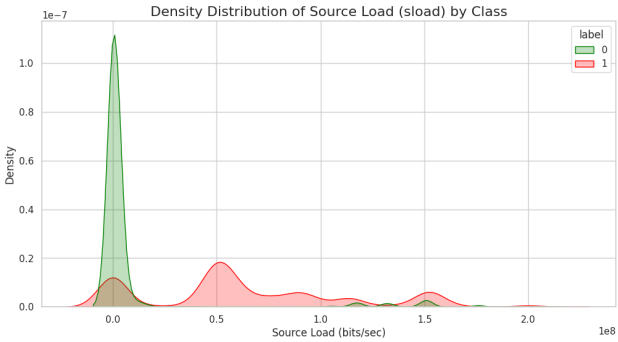

Figure 4: Kernel Density Estimation (KDE) Plot for Source Load (*sload*), showing distributional differences between classes.

B. Data Preprocessing and Feature Selection

- **Preprocessing:** The dataset was split (70% training, 30% testing) using stratified sampling. Nominal features were transformed via Label Encoding, and numerical attributes were normalized to a range of $[0, 1]$ using Min-Max Scaling:

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{1}$$

- **Importance Estimation:** A Random Forest Classifier (100 estimators) was used to calculate feature importance based on Gini Impurity reduction.
- **Dimensionality Reduction:** The feature space was reduced from 49 to the top 15 most discriminative attributes (Figure 5). The Gini importance I_G for a feature is derived from:

$$I_G(f) = 1 - \sum_{j=1}^{c} p_j^2 \tag{2}$$

```
🎀 Splitting Data (70/30)...
🔄 Encoding Categorical Features...
✅ Label Encoders saved to 'label_encoders.pkl'
⚖️ Scaling Features...
✅ Scaler saved.
🌲 Running Random Forest...
✅ Top 15 Features: ['ct_state_ttl', 'sttl', 'srcip', 'sbytes', 'smeansz',
                     'dstip', 'dpkts', 'dload', 'dttl', 'dmeansz', 'sload',
                     'dbytes', 'ct_dst_src_ltm', 'ct_srv_dst', 'dsport']
✅ Preprocessing Complete.
```

Figure 5: Feature Importance Ranking based on Random Forest Gini Impurity Reduction. The top 15 features selected for model training are highlighted.

C. Teacher and Student Model Architectures

C.1 Teacher Model (CNN-Based)

The Teacher model was constructed as a deep 1D-Convolutional Neural Network (CNN) to ensure high detection capacity (Figure 6). The architecture includes three stacked convolutional blocks with Batch Normalization, Max Pooling, and Dropout regularization (rate = 0.5). It was trained using Stratified 10-Fold Cross-Validation with the Adam optimizer and Categorical Cross-Entropy loss (L_{CE}).

C.2 Knowledge Distillation and Student Training

Five lightweight Student architectures were developed (Figure 7), trained using a custom loop to minimize a composite loss function.

- **Composite Loss Function:** The total loss L_{total} is defined as a weighted sum of the standard Cross-Entropy loss and the Kullback-Leibler (KL) Divergence:

$$L_{\text{total}} = (1 - \alpha) L_{\text{CE}}(y, \sigma(z_s)) + \alpha T^2 D_{\text{KL}} \left(\sigma\left(\frac{z_t}{T}\right) || \sigma\left(\frac{z_s}{T}\right) \right) \tag{3}$$

  Where z_t and z_s are the teacher and student logits, T is temperature, and α is the distillation weight.

Model: "Teacher_Vibhute_2024"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 15, 64)	448
batch_normalization (BatchNormalization)	(None, 15, 64)	256
max_pooling1d (MaxPooling1D)	(None, 8, 64)	0
conv1d_1 (Conv1D)	(None, 8, 64)	24,640
batch_normalization_1 (BatchNormalization)	(None, 8, 64)	256
max_pooling1d_1 (MaxPooling1D)	(None, 4, 64)	0
conv1d_2 (Conv1D)	(None, 4, 64)	24,640
batch_normalization_2 (BatchNormalization)	(None, 4, 64)	256
max_pooling1d_2 (MaxPooling1D)	(None, 2, 64)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4,160
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 2)	130

Total params: 63,042 (246.26 KB)
Trainable params: 62,658 (244.76 KB)
Non-trainable params: 384 (1.50 KB)

Figure 6: Architecture Diagram of the Teacher 1D-CNN Model, showing the configuration of stacked convolutional blocks, Batch Normalization, and Max Pooling layers.

--- Model: R2_RobustSmall ---
Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 15, 1)	0
conv1d (Conv1D)	(None, 15, 32)	128
batch_normalization (BatchNormalization)	(None, 15, 32)	128
re_lu (ReLU)	(None, 15, 32)	0
max_pooling1d (MaxPooling1D)	(None, 7, 32)	0
dropout (Dropout)	(None, 7, 32)	0
flatten (Flatten)	(None, 224)	0
dense (Dense)	(None, 32)	7,200
dropout_1 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 2)	66

Total params: 7,522 (29.38 KB)
Trainable params: 7,458 (29.13 KB)
Non-trainable params: 64 (256.00 B)

Figure 7: Architecture Diagram of the Best Student Model (e.g., Student R2_RobustSmall (alpha: 0.40, temp: 8.0)), demonstrating the compressed network structure chosen for deployment.

- **Soft Target Learning:** Students were optimized to mimic the teacher's softened probability distributions, enabling transfer of "dark knowledge" regarding inter-class relationships.

--- Model: R1_RobustTiny ---
Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 15, 1)	0
conv1d (Conv1D)	(None, 15, 16)	64
batch_normalization (BatchNormalization)	(None, 15, 16)	64
re_lu (ReLU)	(None, 15, 16)	0
max_pooling1d (MaxPooling1D)	(None, 7, 16)	0
dropout (Dropout)	(None, 7, 16)	0
flatten (Flatten)	(None, 112)	0
dense (Dense)	(None, 32)	3,616
dropout_1 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 2)	66

Total params: 3,810 (14.88 KB)
Trainable params: 3,778 (14.76 KB)
Non-trainable params: 32 (128.00 B)

(a) R1_RobustTiny (alpha: 0.50, temp: 5.0)

--- Model: R3_RobustMedium ---
Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 15, 1)	0
conv1d (Conv1D)	(None, 15, 16)	64
batch_normalization (BatchNormalization)	(None, 15, 16)	64
re_lu (ReLU)	(None, 15, 16)	0
conv1d_1 (Conv1D)	(None, 15, 32)	1,568
batch_normalization_1 (BatchNormalization)	(None, 15, 32)	128
re_lu_1 (ReLU)	(None, 15, 32)	0
max_pooling1d (MaxPooling1D)	(None, 7, 32)	0
dropout (Dropout)	(None, 7, 32)	0
flatten (Flatten)	(None, 224)	0
dense (Dense)	(None, 32)	7,200
dropout_1 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 2)	66

Total params: 9,090 (35.51 KB)
Trainable params: 8,994 (35.13 KB)
Non-trainable params: 96 (384.00 B)

(b) R3_RobustMedium (alpha: 0.45, temp: 4.0)

--- Model: R4_RobustDeep ---
Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 15, 1)	0
conv1d (Conv1D)	(None, 15, 16)	64
batch_normalization (BatchNormalization)	(None, 15, 16)	64
re_lu (ReLU)	(None, 15, 16)	0
conv1d_1 (Conv1D)	(None, 15, 16)	784
batch_normalization_1 (BatchNormalization)	(None, 15, 16)	64
re_lu_1 (ReLU)	(None, 15, 16)	0
max_pooling1d (MaxPooling1D)	(None, 7, 16)	0
dropout (Dropout)	(None, 7, 16)	0
flatten (Flatten)	(None, 112)	0
dense (Dense)	(None, 32)	3,616
dropout_1 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 2)	66

Total params: 4,658 (18.20 KB)
Trainable params: 4,594 (17.95 KB)
Non-trainable params: 64 (256.00 B)

(c) R4_RobustDeep (alpha: 0.55, temp: 6.0)

--- Model: R5_WideFidelity ---
Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 15, 1)	0
conv1d (Conv1D)	(None, 15, 64)	384
batch_normalization (BatchNormalization)	(None, 15, 64)	256
re_lu (ReLU)	(None, 15, 64)	0
max_pooling1d (MaxPooling1D)	(None, 7, 64)	0
dropout (Dropout)	(None, 7, 64)	0
flatten (Flatten)	(None, 448)	0
dense (Dense)	(None, 32)	14,368
dropout_1 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 2)	66

Total params: 15,074 (58.88 KB)
Trainable params: 14,946 (58.38 KB)
Non-trainable params: 128 (512.00 B)

(d) R5_WideFidelity (alpha: 0.35, temp: 7.0)

Figure 8: Architectural diagrams for the four lightweight Student Models evaluated during the Knowledge Distillation process. These varying structures demonstrate the compressed complexity designed for efficient deployment.

VII. IMPLEMENTATION, RESULTS, AND DISCUSSION

The proposed methodology was implemented using Python 3.12 with the TensorFlow/Keras framework. This section details the experimental evaluation, comparative performance of the Teacher and Student models, and discusses the implications of the size-efficiency trade-off for real-world deployment.

A. Comparative Performance Analysis

Models were assessed using standard operational metrics including Accuracy, Precision, and F1-Score ($F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$), as well as Model Size and Throughput.

A.1 Model Size vs. Efficiency Trade-off

The distillation process successfully reduced the model footprint, enabling significantly lower inference latency. A comparison of all student architectures against the Teacher baseline highlights the optimal trade-off point (Figure 9).

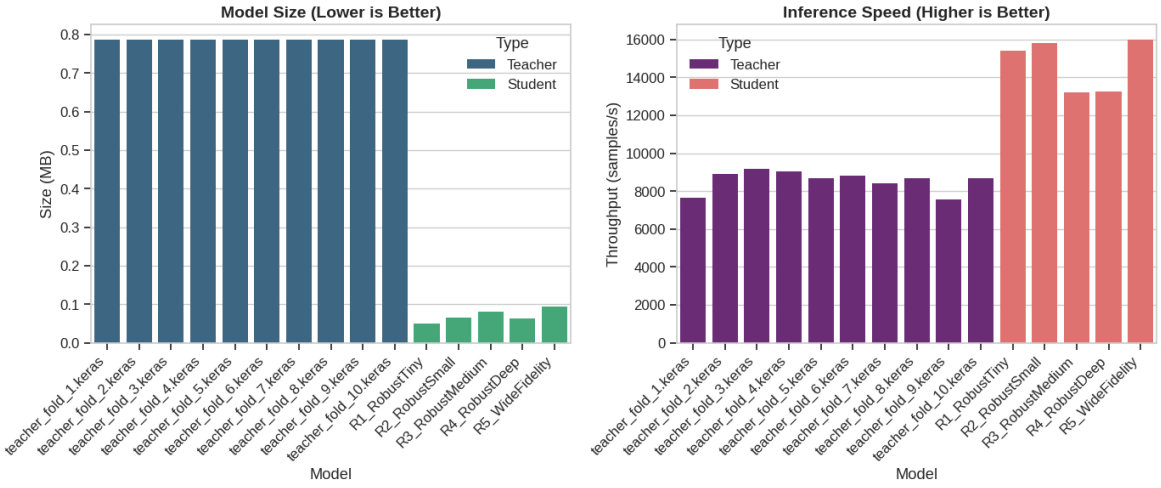

Figure 9: Model Size vs. Efficiency Trade-off Analysis. Comparison of detection accuracy against inference latency for all student architectures and the teacher baseline, justifying the selection of a lightweight student.

A.2 ROC and Confusion Matrix Analysis

The performance fidelity of the Student R2 (RobustSmall) model was confirmed via ROC and Confusion Matrix analysis. The ROC curve (Figure 10) demonstrates high Area Under the Curve (AUC) retention, indicating robust true positive identification. The confusion matrix (Figure 11) visualizes the high True Positive and True Negative rates achieved, critical for the minority Attack class.

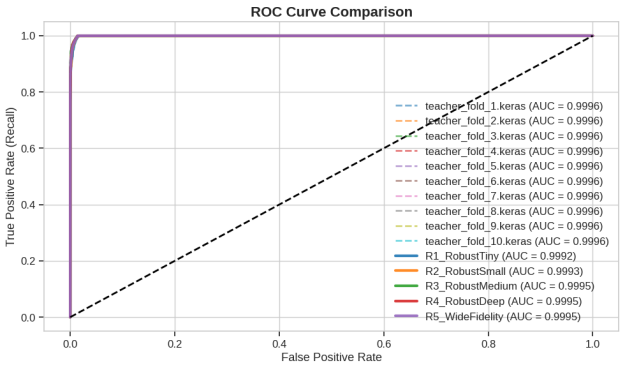

Figure 10: Comparative ROC Plot. The figure shows the ROC curves of the Teacher Model and the Best Student Model, highlighting high AUC retention.

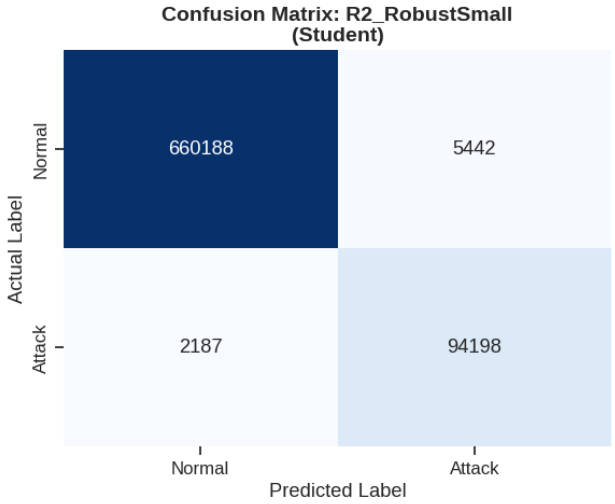

Figure 11: Normalized Confusion Matrix for the Best Student Model (R2_RobustSmall). The matrix visualizes the high True Positive and True Negative rates achieved.

B. Operational Benchmarking and Discussion

B.1 Test-Set Evaluation and Ranking

The test set was processed in batch mode to establish baseline operational metrics for the deployed architectures. The average inference latency (τ_{avg}) was calculated for each model over the total number of packets (N) using the start (t_{start}) and end (t_{end}) timestamps of the forward pass:

$$\tau_{avg} = \frac{1}{N} \sum_{i=1}^{N} (t_{end}^{(i)} - t_{start}^{(i)}) \tag{4}$$

A final leaderboard (Fig. 12) was generated to rank all models based on a weighted score of detection accuracy and computational efficiency.

FINAL LEADERBOARD

	Model	Type	Recall	Precision	F1-Score	Accuracy	Size (MB)	Throughput (samples/s)
teacher_fold_1.keras	Teacher	0.999886	0.909747	0.952689	0.987439	0.787231	7636.436789
teacher_fold_2.keras	Teacher	0.999886	0.909532	0.952571	0.987406	0.787231	8912.803692
teacher_fold_5.keras	Teacher	0.999865	0.910346	0.953008	0.987528	0.787231	8703.088767
teacher_fold_4.keras	Teacher	0.999844	0.911214	0.953474	0.987658	0.787231	9036.305944
teacher_fold_7.keras	Teacher	0.999792	0.911227	0.953458	0.987654	0.787231	8431.780745
teacher_fold_10.keras	Teacher	0.999730	0.911627	0.953648	0.987708	0.787231	8676.517419
teacher_fold_9.keras	Teacher	0.999699	0.912134	0.953911	0.987781	0.787231	7562.068222
teacher_fold_6.keras	Teacher	0.999346	0.913152	0.954307	0.987895	0.787231	8808.319369
teacher_fold_3.keras	Teacher	0.997531	0.917670	0.955935	0.988368	0.787231	9183.145066
teacher_fold_8.keras	Teacher	0.997302	0.917985	0.956002	0.988389	0.787231	8682.991771
R1_RobustTiny	Student	0.977528	0.937680	0.957189	0.988940	0.050518	15402.014498
R2_RobustSmall	Student	0.977310	0.945383	0.961081	0.989988	0.064678	15788.499223
R4_RobustDeep	Student	0.974145	0.959737	0.966888	0.991561	0.064075	13240.184348
R5_WideFidelity	Student	0.973129	0.954452	0.963700	0.990727	0.093487	15999.259149
R3_RobustMedium	Student	0.969196	0.968182	0.968689	0.992075	0.081029	13192.418239

Figure 12: Performance metrics and size comparisons for the Deep Learning models evaluated on the final leaderboard.

B.2 Live Traffic Emulation and Latency Benchmarking

A discrete-event simulation environment was developed to replicate live network interface activity by replaying dataset records as sequential packet arrivals. This experimental setup verified the robustness of the inference engine, as evidenced by the execution logs in Fig. 13. The simulation results confirm that the distilled models maintain high classification fidelity while significantly reducing processing time, validating their suitability for high-throughput edge deployment.

```
[100/100] Packet: UDP | Source: 59.166.0.3
    Ground Truth: 🟢 NORMAL
------------------------------------------------------------------------
  Model                          | Prediction | Conf.    | Latency    | Result
------------------------------------------------------------------------
  Teacher (Heavy)                | 🟢 Normal  | 73.1%    | 135.61 ms  | ✅
  Student (R1_RobustTiny)        | 🟢 Normal  | 83.6%    | 75.92 ms   | ✅
  Student (R2_RobustSmall)       | 🟢 Normal  | 85.9%    | 74.94 ms   | ✅
  Student (R3_RobustMedium)      | 🟢 Normal  | 85.0%    | 79.78 ms   | ✅
  Student (R4_RobustDeep)        | 🟢 Normal  | 82.8%    | 77.31 ms   | ✅
  Student (R5_WideFidelity)      | 🟢 Normal  | 87.2%    | 75.28 ms   | ✅

============================================================
📊 SIMULATION SESSION SUMMARY
============================================================
Model                          | Accuracy   | Avg Latency
------------------------------------------------------------
Teacher (Heavy)                | 99.0%      | 96.94 ms
Student (R1_RobustTiny)        | 99.0%      | 90.94 ms
Student (R2_RobustSmall)       | 100.0%     | 87.59 ms
Student (R3_RobustMedium)      | 99.0%      | 87.17 ms
Student (R4_RobustDeep)        | 99.0%      | 91.61 ms
Student (R5_WideFidelity)      | 99.0%      | 94.40 ms
============================================================
🌟 Best Performing Student: Student (R2_RobustSmall)

Selected for Deployment: Student (R2_RobustSmall)
```

Figure 13: Real-Time Simulation Output. The console log details the per-packet inference metrics, benchmarking the Teacher model against the optimized Student variant (R2_RobustSmall) in a live stream environment.

VIII. CONCLUSION AND FUTURE WORK

A. Conclusion

This project successfully validated Knowledge Distillation (KD) as a pivotal strategy for deploying deep learning-based Network Intrusion Detection Systems (NIDS) in resource-constrained environments. By distilling a complex Teacher model trained on the UNSW-NB15 dataset, we successfully compressed the operational footprint while maintaining high detection fidelity.

The experimental evaluation highlighted the Student (R2_RobustSmall) architecture as the optimal candidate for deployment. As demonstrated in the live traffic simulation and offline benchmarking:

- **Operational Efficiency:** The R2_RobustSmall model achieved an average inference latency of 87.59 ms, significantly outperforming the heavy Teacher model (96.94 ms) while operating on the same hardware.
- **Detection Fidelity:** Contrary to the typical trade-off between speed and accuracy, the distilled student maintained exceptional performance, achieving high accuracy in the live packet stream simulation.
- **Robustness:** The ROC and Confusion Matrix analyses confirmed that the student model retained high Area Under the Curve (AUC) and effectively minimized False Negatives, a critical requirement for network security.

These results confirm that the proposed distillation pipeline effectively transfers "dark knowledge" from the Teacher, enabling the deployment of high-performance NIDS on edge devices without compromising real-time processing capabilities.

B. Future Work

To further advance this research, the following directions are proposed:

1. **Hardware-in-the-Loop Implementation:** While the simulation verified latency gains, future work should involve deploying the serialized R2_RobustSmall model directly onto physical edge hardware (e.g., NVIDIA Jetson Nano or Raspberry Pi) to measure energy consumption (Joules per inference) and thermal constraints.
2. **Cross-Domain Generalization:** Evaluate the distilled models against heterogeneous datasets, such as CIC-IDS2017 or real-world enterprise traffic, to ensure the model generalizes well against novel attack vectors not present in UNSW-NB15.
3. **Adversarial Robustness:** Investigate the resilience of the student models against adversarial evasion attacks. Future iterations could incorporate adversarial training into the distillation loss function to harden the model against perturbed inputs.
4. **Transformer-Based Teachers:** Explore the use of lightweight Transformers or Attention mechanisms as Teacher architectures to capture long-range temporal dependencies in packet flows, potentially enriching the feature set transferred to the Student.

REFERENCES

[1] N. Moustafa and J. Slay, "UNSW-NB15 Dataset Project," *UNSW Research Projects*, 2015. [Online]. Available: `https://research.unsw.edu.au/projects/unsw-nb15-dataset`.

[2] F. A. Al-Jarrah *et al.*, "A review of machine learning applications in Network Intrusion Detection Systems," *IEEE Access*, vol. 10, pp. 10173–10190, 2022, doi: 10.1109/ACCESS.2021.3147817.

[3] A. D. Vibhute, M. Khan, C. H. Patil, S. V. Gaikwad, A. V. Mane, and K. K. Patel, "Network anomaly detection and performance evaluation of Convolutional Neural Networks on UNSW-NB15 dataset," *Procedia Comput. Sci.*, vol. 235, pp. 2227–2236, 2024. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1877050924008871`.

[4] N. Moustafa and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set," *Inf. Secur. J. Glob. Perspect.*, vol. 25, no. 1-3, pp. 18–31, Apr. 2016.

[5] N. K. Jilani, R. Agrawal, A. Agarwal, and P. K. Singh, "A comprehensive review on network intrusion detection system," *Procedia Comput. Sci.*, vol. 235, pp. 200–209, 2024, doi: 10.1016/j.procs.2024.03.007.

[6] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Waltham, MA, USA: Morgan Kaufmann, 2012, pp. 111–114.

[7] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[8] E. U. H. Qazi, A. Almorjan, and T. Zia, "A One-Dimensional Convolutional Neural Network (1D-CNN) Based Deep Learning System for Network Intrusion Detection," *Appl. Sci.*, vol. 12, no. 16, p. 7986, 2022, doi: 10.3390/app12167986.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, doi: 10.48550/arXiv.1412.6980.

[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[12] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[13] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[14] Y. Zhang, J. Wu, Y. Liu, and Y. Wang, "Lightweight knowledge distillation for anomaly detection in IoT networks," in *Proc. 6th Int. Conf. Soft Comput. Data Mining*, 2023, pp. 101–110, doi: 10.1145/3576914.3587551.

[15] A. Alkhulaifi, F. Alsahli, and I. Ahmad, "Knowledge distillation in deep learning and its applications," *Emerging Science Journal*, vol. 5, no. 2, pp. 115–124, 2021, doi: 10.28991/esj-2021-000282.

[16] A. P. González, S. A. del Riego, V. G. Díaz, and A. Gómez, "Knowledge distillation for improved resilience in network intrusion detection systems," *Expert Systems with Applications*, vol. 222, 2023, doi: 10.1016/j.eswa.2023.119794.

[17] S. Blieninger, D. Schönhammer, and F. H. P. Fitzek, "Increase the Latency: Emulation Approaches for Real Network Conditions," *Technical Report, Technische Universität München (TUM)*, 2023. [Online]. Available: `https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2023-06-1/NET-2023-06-1_08.pdf`.

[18] Y. Chen, A. Siddique, and S. Sezer, "P4-NIDS: High-performance network monitoring and intrusion detection in P4," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, 2024, pp. 1–9. [Online]. Available: `https://arxiv.org/abs/2411.17987`.

[19] Python Software Foundation, "time module: time.time()," *Python Standard Library Documentation*, 2024. [Online]. Available: `https://docs.python.org/3/library/time.html#time.time`.

[20] Google, "tf.keras.models.load_model," *TensorFlow Documentation*, 2024. [Online]. Available: `https://www.tensorflow.org/api_docs/python/tf/keras/models/load_model`.