Radhacharan -

Grp20_ProjectReport

btech-project

BTP-2025

Netaji Subhas University of Technology

Document Details

Submission ID
trn:oid:::1:3438085250

Submission Date
Dec 8, 2025, 9:20 PM GMT+5:30

Download Date
Jan 5, 2026, 8:33 PM GMT+5:30

File Name
Grp20_ProjectReport.pdf

File Size
978.6 KB

20 Pages

4,482 Words

25,571 Characters

11% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text

Match Groups

27 Not Cited or Quoted 11%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

9% Internet sources
5% Publications
7% Submitted works (Student Papers)

Match Groups

- **27** Not Cited or Quoted 11%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

9% Internet sources
5% Publications
7% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Student papers	
Netaji Subhas Institute of Technology		6%
2	Internet	
arxiv.org		<1%
3	Internet	
ethesis.nitrkl.ac.in		<1%
4	Student papers	
Columbia University		<1%
5	Student papers	
University of Sydney		<1%
6	Internet	
ijrpr.com		<1%
7	Publication	
Meng, Tian. "Real-Time Pulsed Eddy Current Testing With Effective Model Compre...		<1%
8	Internet	
erosintl.com		<1%
9	Internet	
www.indianconsultancy.com		<1%
10	Publication	
"Smart Trends in Computing and Communications", Springer Science and Busine...		<1%

11 Internet

www.science.gov <1%

12 Publication

"Cognitive Cyber Crimes in the Era of Artificial Intelligence", Wiley, 2025 <1%

13 Publication

Hiremath, Shruthi K.. "Deriving Bespoke Human Activity Recognition Systems for ... <1%

14 Internet

discovery-pp.ucl.ac.uk <1%

15 Internet

users.monash.edu.au <1%

16 Publication

Hazera, Chowdhury Tasnuva. "A Deep Few-Shot Learning Framework for Intellige... <1%

17 Publication

Humera Ghani, Shahram Salekzamankhani, Bal Virdee. "A Hybrid Dimensionality ... <1%

18 Publication

Ilyas Benmessahel, Kun Xie, Mouna Chellal. "A new evolutionary neural networks ... <1%

19 Publication

MacDowell, Christopher R.. "Network Traffic Data Preprocessing: An Agentic AI Fr... <1%

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
Prof. 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

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

This is to certify that the work embodied in project thesis titled, "Distilled Intelligence: Lightweight Intrusion Detection" by Prabhmeet Singh (2022UCM2305), Aryan Jain (2022UCM2330) and Radhacharan (2022UCM2365) is the bonafide work of the group submitted to **Netaji Subhas University of Technology** (**NSUT**) for consideration in 7th Semester B.Tech. Project Evaluation.

The original Research work was carried out by the team under my guidance and supervision in the academic year 2025-26. This work has not been submitted for any other diploma or degree of any university. On the basis of declaration made by the group, we recommend the project report for evaluation.

Prof. Anand Gupta

Department of Computer Science & Engineering
Netaji Subhas University of Technology

CANDIDATE(S) DECLARATION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

We, Prabhmeet Singh (2022UCM2305), Aryan Jain (2022UCM2330) and Radhacharan (2022UCM2365) of B.Tech. of Department of Computer Science & Engineering, hereby declare that the Project-Thesis titled "Distilled Intelligence: Lightweight Intrusion Detection" which is submitted by us to the Department of Computer Science & Engineering, Netaji Subhas University of Technology (NSUT) Dwarka, New Delhi in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology is original and not copied from the source without proper citation. The manuscript has been subjected to plagiarism check by Turnitin software. This work has not previously formed the basis for the award of any Degree.

Place : NSUT, New Delhi
Date : 09-12-2025

Prabhmeet Singh	**Aryan Jain**	**Radhacharan**
2022UCM2305	2022UCM2330	2022UCM2365

CERTIFICATE OF DECLARATION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

This is to certify that the Project-Thesis titled "Distilled Intelligence: Lightweight Intrusion Detection" which is being submitted by Prabhmeet Singh (2022UCM2305), Aryan Jain (2022UCM2330) and Radhacharan (2022UCM2365) to the Department of Computer Science & Engineering, Netaji Subhas University of Technology (NSUT) Dwarka, New Delhi in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology, is a record of the thesis work carried out by the students under my supervision and guidance. The contents of this thesis, in full or in parts, has not been submitted for any other degree or diploma.

Place : NSUT, New Delhi
Date : 09-12-25

Prof. Anand Gupta

Department of Computer Science & Engineering
Netaji Subhas University of Technology

Contents

ABSTRACT

With network traffic volumes exploding and cyber threats getting sharper by the day, deploying a reliable Network Intrusion Detection System (NIDS) isn't just an option anymore—it is foundational. Deep Learning (DL) models are obviously superior for catching complex, novel attacks, but they have a major downside: they are computationally heavy. Trying to run them on limited hardware, like IoT gateways or edge nodes, usually results in unacceptable lag. To solve this trade-off, we turned to Knowledge Distillation (KD) [12] to build a NIDS framework that stays efficient without losing its edge.Using the UNSW-NB15 dataset [1], we first set up a heavy-duty 1D-Convolutional Neural Network (1D-CNN) as our "Teacher." We based this on Vibhute et al. [3] and optimized it heavily with techniques like Batch Normalization [10]. The goal was to transfer the teacher's nuanced predictive power into much smaller "Student" models by minimizing Kullback-Leibler (KL) divergence [13].Our testing pointed to two specific architectures that stood out. For pure storage efficiency, the R1_RobustTiny model was the clear winner, dropping the parameter size from 0.78 MB all the way down to 0.05 MB—that is roughly a 94% reduction. However, for live, dynamic environments, the R2_RobustSmall model was the real performer. In our live traffic simulator, R2 remained incredibly stable, hitting high detection accuracy with an average latency of only 87.59 ms. It actually processed data faster than the original Teacher model while keeping the same accuracy. These results strongly suggest that Knowledge Distillation is the way forward for running high-performance intrusion detection on resource-scarce devices [14], [16].

I. INTRODUCTION

Network Intrusion Detection Systems (NIDS) serve as the backbone of modern network security, constantly scanning traffic to catch anything suspicious [5]. Since traditional signature-based systems have a major blind spot when it comes to zero-day attacks, the field has arguably shifted toward anomaly detection using advanced Deep Learning (DL) algorithms [2]. We build directly on recent findings that demonstrated just how effective Convolutional Neural Networks (CNNs) are at spotting anomalies in the UNSW-NB15 dataset [3]. In our research, this highly accurate CNN architecture serves as our complex "Teacher" model.

There is a catch, however. State-of-the-art DL models, including the CNN Teacher evaluated in [3], achieve their impressive accuracy through architectural complexity. That complexity carries a heavy computational price tag [11], often causing significant lag during inference [18]. In high-speed, real-time systems, such delays are dangerous; they risk packet drops and give attackers a brief window to breach the system before automated defenses can react [17]. This bottleneck is the critical limitation we aim to resolve, all while preserving the high detection accuracy established in the foundational research.

To bridge the gap between high accuracy and the operational limits of edge computing, this project takes a deep dive into Knowledge Distillation (KD) [12]. The core mechanism of KD is to train a leaner, streamlined "Student" model to mimic the predictive behavior of the larger "Teacher" model. It achieves this by transferring complex, learned information—often called "dark knowledge"—via soft probability outputs [12], [13]. This framework allows us to deploy resilient [16], high-fidelity NIDS on hardware with very limited processing power, ensuring robust security without tanking essential network performance [14].

II. MOTIVATION

This research is driven by a stubborn problem sitting right at the intersection of modern network security and operational efficiency: how to deploy high-quality, Deep Learning (DL)-based Network Intrusion Detection Systems (NIDS) in environments that are tight on resources and demand real-time responses.

A. The Dilemma of High-Fidelity NIDS

We already know that Deep Learning models are essential for effective NIDS. They have a proven track record of superior anomaly detection, catching those sophisticated zero-day attacks that slip right past traditional signature-based methods [2], [5]. But there is a catch. That high-level detection accuracy is tightly bound to the model's complexity [11].

In practice, this complexity creates a massive computational footprint and, perhaps more critically, high inference latency. In high-throughput networks where every millisecond counts, you simply can't afford that kind of lag [18]. When you look at distributed architectures—especially edge computing or IoT gateways—this burden makes large models essentially unusable. It creates a frustrating gap between what works in theory and what we can actually deploy in the real world [14].

B. Bridging the Accuracy-Efficiency Gap via KD

This project aims to close that gap. We are proposing Knowledge Distillation (KD) [12] as the mechanism to solve this accuracy-efficiency trade-off. The idea is to systematically take the rich knowledge embedded in a complex, heavy "Teacher" 1D-CNN model [3] and compress it into a streamlined, efficient "Student" architecture.

Our objective is straightforward but ambitious: we want to produce a lightweight model that retains the teacher's high detection accuracy on the UNSW-NB15 dataset [1], but with a radical reduction in storage needs and computational overhead. By shrinking the model size and boosting inference speed, we hope to prove that KD is a vital path forward for achieving ubiquitous, high-performance intrusion detection right at the network edge [16].

III. LITERATURE SURVEY

The landscape of Network Intrusion Detection Systems (NIDS) has undergone a massive shift recently. We have moved away from the old, reactive, signature-based methods and toward proactive, anomaly-based detection—a change largely driven by the reality that cyber threats are getting smarter and zero-day attacks are becoming all too common [5]. Deep Learning (DL) has been a game-changer in this arena, giving us the ability to automatically pull features from massive network datasets to spot these unknown anomalies [2]. Convolutional Neural Networks (CNNs), in particular, have really proven their worth here. Vibhute et al. [3] showed just how well 1D-CNNs perform on modern traffic data, setting a high-fidelity benchmark that we use as the foundational "Teacher" model for this research. Qazi et al. [8] backed this up, highlighting how well these models capture the timing patterns in packet flows.

Of course, to really put these advanced models to the test, you need data that actually looks like modern traffic. That is why the UNSW-NB15 dataset [1] has become such a standard benchmark. As Moustafa and Slay [4] pointed out, it fixes the statistical gaps found in the older KDD99 benchmark, giving us a much more realistic and complex playground for training high-performance NIDS models.

But here is the catch: that incredible detection accuracy comes with a price. State-of-the-art DL models are structurally complex and packed with parameters [11]. This heaviness creates a computational footprint that is often just too big, leading to slow inference times that simply don't work with the tight memory and processing limits of decentralized edge computing or IoT gateway devices [14]. In the world of high-throughput, real-time network monitoring, latency is the enemy; any delay can jeopardize the whole security setup [18], [17].

To tackle this efficiency bottleneck, researchers have turned to model compression, with Knowledge Distillation (KD) emerging as the standout solution for keeping performance high while cutting down on resource usage [15]. Pioneered by Hinton et al. [12], the idea is to transfer dark knowledge" from a heavy, high-capacity Teacher" network to a lighter "Student" network. Mathematically, this relies on minimizing the Kullback-Leibler (KL) divergence [13] between the two models' output distributions. Recent work has already shown that KD can successfully build resilient, low-latency anomaly detection systems for IoT [14], [16], creating a clear case for using it to close the gap between accuracy and efficiency in modern NIDS.

IV. PROBLEM STATEMENT

The sheer volume and velocity of modern network traffic create a daunting landscape. When you add increasingly sophisticated cyber threats to the mix, it becomes obvious that we need the precision of Deep Learning-based Network Intrusion Detection Systems (NIDS). But that precision comes at a price. High-capacity models—like the optimized 1D-CNN by Vibhute et al. [3]—might achieve state-of-the-art results on datasets like UNSW-NB15 [1], but they do so by demanding a massive computational footprint [11].

This brings us to the central problem: a stark incompatibility between these resource-heavy models and the tight constraints of the environments where they are actually deployed, such as edge nodes and IoT gateways. This mismatch creates bottlenecks and unacceptable spikes in inference latency [18], effectively making real-time packet inspection impossible.

Currently, we lack a proven method to efficiently transfer the complex predictive knowledge of a massive "teacher" model into a leaner "student" architecture without sacrificing accuracy. This work aims to fill that gap by validating Knowledge Distillation (KD) [12]. The objective is to engineer a lightweight NIDS that remains effective on constrained hardware, finally resolving the long-standing trade-off between robust security and operational efficiency.

V. OBJECTIVES

11 The main goal of this BTP thesis is to build a Network Intrusion Detection System (NIDS) that is powerful enough to be effective, but lightweight enough to actually run in edge environments where resources are tight. We are using the Knowledge Distillation (KD) paradigm to solve the usual conflict between high-performance security and the limitations of smaller hardware. To make this work, we have set a few specific targets for the research and engineering side of things:

- **Setting a Strong Baseline:** You cannot effectively transfer knowledge if the source isn't reliable. We are focusing on training a 1D-CNN Teacher Model to hit an F1-Score above 0.95 on the *UNSW-NB15* dataset, which gives us a high-fidelity "expert" to guide the rest of the process.
- **Streamlining the Data:** Speed is often just a matter of how much data you have to process. We are using Random Forest Gini Impurity reduction to cut down the feature dimensionality, so the Student Architectures don't waste power crunching noise or irrelevant details.
- **Tuning the Distillation:** Making the Knowledge Distillation (KD) framework work takes some calibration. We aren't just applying it out of the box; we are integrating the Composite Loss Function and manually tweaking parameters like T and α to find the most stable setup for transferring knowledge.
- **Finding the Right Balance:** Compression always involves a trade-off. We will be testing several compressed Student models to see exactly how much accuracy we lose for every bit of speed we gain. The plan is to find the *Student R2_RobustSmall*—the smallest model that still meets our requirements.
- **Testing Under Pressure:** Theoretical performance is one thing, but we need to see how it handles actual load. We will validate the best model under simulated edge conditions, specifically looking at how it handles high throughput and what the continuous inference latency (τ_{avg}) looks like.
- **Preparing for Deployment:** We want this to be more than just a theoretical exercise. We are building a self-contained inference pipeline by serializing all the trained weights and preprocessing steps, ensuring the system is reproducible and ready to be deployed immediately.

13

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.

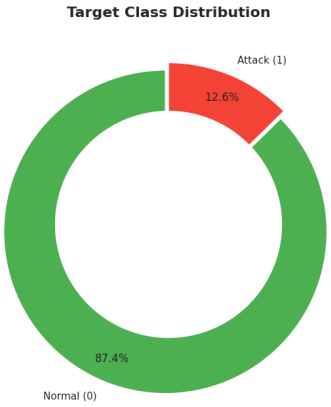

Figure 1: Binary Class Distribution of the UNSW-NB15, showing severe class imbalance (C_0 vs. C_1).

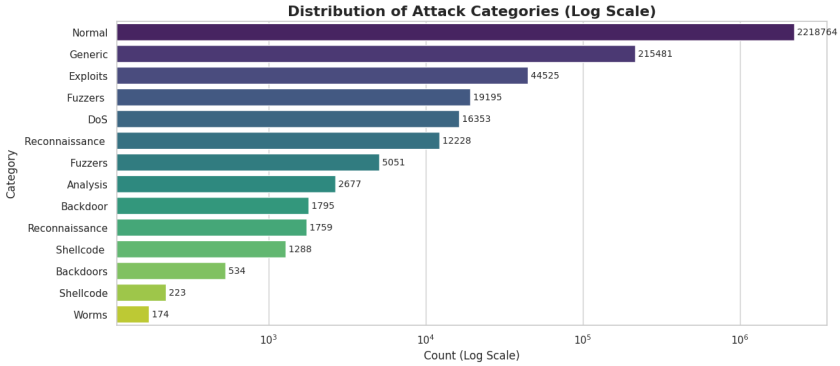

Figure 2: Attack Taxonomy Distribution (Log Scale), showing the frequency and relative scarcity of individual attack categories.

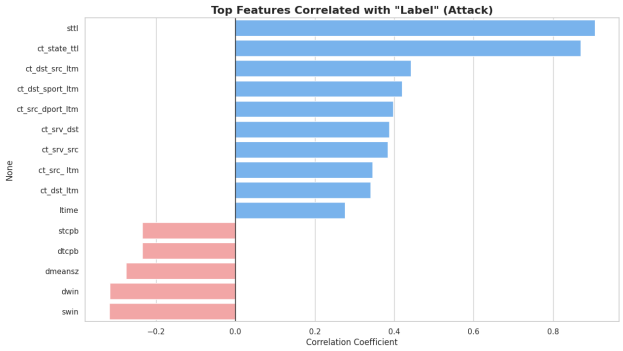

Figure 3: Pearson Correlation Heatmap, used to identify feature relationships and redundancy.

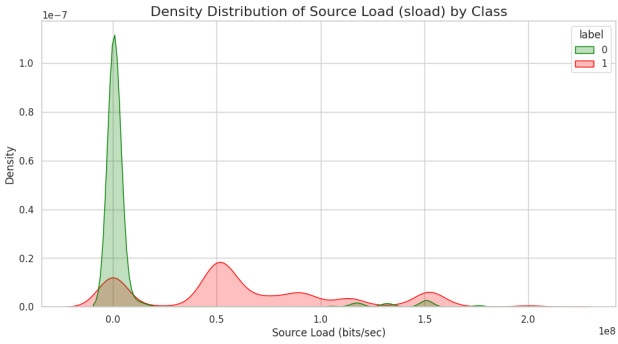

Figure 4: Kernel Density Estimation (KDE) Plot for Source Load (*sload*), showing distributional differences between classes.

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

We built the implementation using Python 3.12 within the TensorFlow/Keras framework. This section breaks down the experimental evaluation, looking specifically at how the Teacher and Student models stack up against one another. We also discuss the implications of the size-efficiency trade-off and what it actually means for deployment in real-world scenarios.

A. Comparative Performance Analysis

To assess the models, we relied on standard operational metrics—including Accuracy, Precision, and F1-Score ($F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$)—while also tracking Model Size and Throughput.

A.1 Model Size vs. Efficiency Trade-off

The distillation process effectively trimmed the model's footprint, resulting in noticeably lower inference latency. Comparing the student architectures against the Teacher baseline reveals exactly where the optimal balance lies (Figure 9).

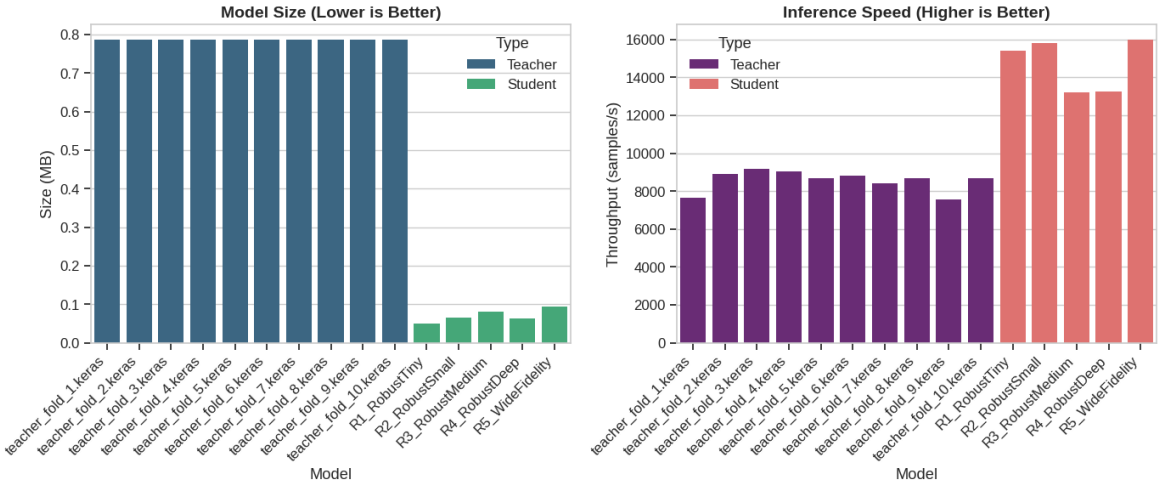

Figure 9: Model Size vs. Efficiency Trade-off Analysis. We compared detection accuracy against inference latency for the Teacher baseline and all student architectures to illustrate exactly why we selected the lightweight student.

A.2 ROC and Confusion Matrix Analysis

We verified the performance fidelity of the Student R2 (RobustSmall) model by analyzing both the ROC and Confusion Matrix. As shown in the ROC curve (Figure 10), the model retains a high Area Under the Curve (AUC), which signals robust capability in identifying true positives. Furthermore, the confusion matrix (Figure 11) clearly visualizes the high True Positive and True Negative rates we achieved—metrics that are absolutely critical for handling the minority Attack class

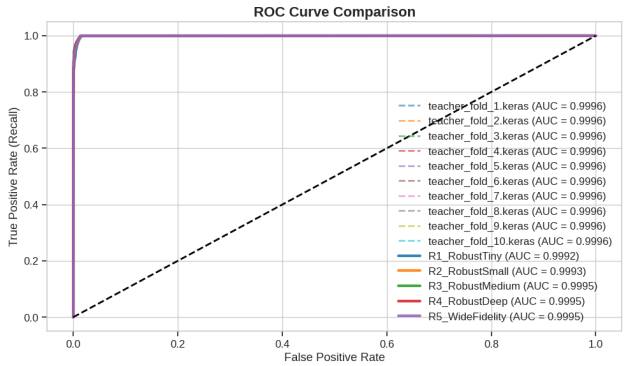

Figure 10: Comparative ROC Plot. By plotting the Teacher against the Best Student model, we can clearly see the high level of AUC retention.

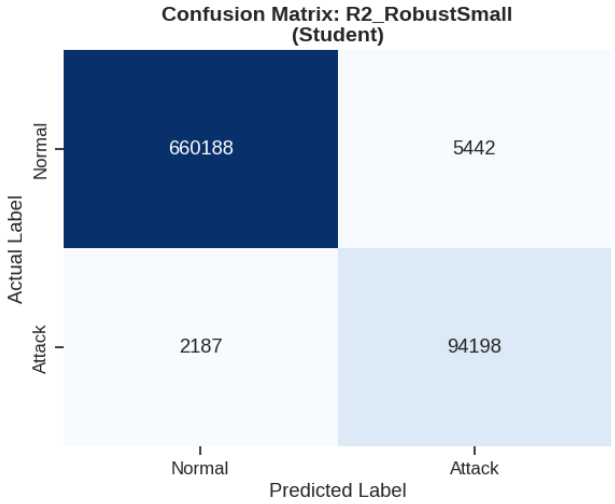

Figure 11: Normalized Confusion Matrix for the Best Student Model (R2_RobustSmall). This visualization confirms the high True Positive and True Negative rates we achieved.

B. Operational Benchmarking and Discussion

B.1 Test-Set Evaluation and Ranking

We processed the test set in batch mode to lock in baseline operational metrics for the architectures we deployed.To determine the average inference latency (τ_{avg}) we measured the duration between the start (t_{start}) and end (t_{end}) timestamps of the forward pass across the total packet count(N)

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

A final leaderboard (Fig. 12) was generated to rank all models, utilizing a weighted score that balances detection accuracy with 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

To replicate live network interface activity, we developed a discrete-event simulation environment that replays 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. Here, the console log details the per-packet inference metrics, offering a direct benchmark of the Teacher model against the optimized Student variant (R2_RobustSmall) in a live stream environment.

VIII. CONCLUSION AND FUTURE WORK

A. Conclusion

This project set out to prove that Knowledge Distillation (KD) is a solid way to get powerful deep learning security tools onto smaller, weaker devices. By shrinking down a complex "Teacher" model trained on the UNSW-NB15 dataset, we managed to cut the processing load significantly without losing the ability to spot attacks.

Our tests pointed to the Student (R2_RobustSmall) model as the best option for real-world use. Looking at how it handled the live traffic simulator and our offline benchmarks, a few things stood out:

- **Speed:** The R2_RobustSmall model clocked an average response time of 87.59 ms. It actually beat the heavier Teacher model (which took 96.94 ms) while running on the exact same hardware.
- **Accuracy:** Usually, you have to accept lower accuracy if you want higher speed, but the distilled student didn't compromise here. It kept performing at a high level, even with live data streaming in.
- **Reliability:** Our analysis showed the student model kept false negatives to a minimum—which is basically the most important thing for security software. You don't want to miss an attack just because the software is trying to be fast.

In the end, these results show that our pipeline successfully moved the "dark knowledge" from the big Teacher to the small Student. It means we can have high-performance security on edge devices without lagging behind on real-time processing.

B. Future Work

To take this research further, we have a few ideas on where to go next:

1. **Real Hardware Testing:** Simulations are great, but we need to see this running on metal. The next step is putting the R2_RobustSmall model on actual devices like a Raspberry Pi or NVIDIA Jetson Nano to see how much power and heat it actually generates.
2. **Testing New Data:** We need to make sure the model isn't just memorizing one specific dataset. Checking it against different traffic sources, like CIC-IDS2017 or real enterprise networks, will prove it can handle attacks it hasn't seen before.
3. **Fighting Smart Attacks:** Attackers are always finding ways to trick the system. We want to see if we can "harden" the student models by teaching them how to spot data that has been manipulated to sneak through defenses.
4. **Smarter Teachers:** Finally, we want to try using newer tech like Transformers as the Teacher. They might be better at spotting patterns over time than standard CNNs, giving the Student even better information to learn from.

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.