B.Tech. BCSE497J - Project-I

A COMPARATIVE STUDY OF DEEP LEARNING MODELS FOR IOT INTRUSION DETECTION

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

22BCI0034 Vibhor Puri

22BCE0050 Aishi Adhikari

Under the Supervision of

Dr. SARITHA MURALI

Assistant Professor Sr. Grade 1

Department of Database Systems

School of Computer Science and Engineering (SCOPE)

November 2025

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

DECLARATION

I hereby declare that the project entitled "**A COMPARATIVE STUDY OF DEEP LEARNING MODELS FOR IOT INTRUSION DETECTION**" submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Saritha Murali.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 5/11/2025

Aishi Adhikari

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled "**A COMPARATIVE STUDY OF DEEP LEARNING MODELS FOR IOT INTRUSION DETECTION**" submitted by Aishi Adhikari (22BCE0050), **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by her under my supervision during Fall Semester 2025-2026, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 5/11/2025

12/11/2025

Signature of the Guide

12/11/2025 12/11/25

Examiner(s)

Dr. Boominathan P

Bachelor of Technology in Computer Science and Engineering

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT has enabled me to explore and develop our ideas to their fullest potential.

My sincere thanks to Dr. Jaisankar N, the Dean of the School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence. The Dean's dedication to academic excellence and innovation has been a constant source of motivation for me. I appreciate his efforts in creating an environment that nurtures creativity and critical thinking.

I express our profound appreciation to Dr. Boominathan P, the Head of Department of Software Systems, for his insightful guidance and continuous support. His expertise and advice have been crucial in shaping the direction of my project. The Head of Department's commitment to fostering a collaborative and supportive atmosphere has greatly enhanced my learning experience. His constructive feedback and encouragement have been invaluable in overcoming challenges and achieving my project goals.

I am immensely thankful to our project guide, Dr. Saritha Murali, for her dedicated mentorship and invaluable feedback. Her patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor's willingness to share her expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. Her support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

Name of the Candidate
Aishi Adhikari

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
AUC	Area Under the Curve
CNN	Convolutional Neural Network
CSV	Comma Separated Values
DDoS	Distributed Denial of Service
DL	Deep Learning
DoS	Denial of Service
GPU	Graphics Processing Unit
IDS	Intrusion Detection Systems
LSTM	Long Short-Term Memory
ML	Machine Learning
ROC	Receiver Operating Characteristic
SHAP	SHapley Additive exPlanations
SMOTE	Synthetic Minority Oversampling Technique
SOC	Security Operations Centers
XAI	Explainable Artificial Intelligence
XSS	Cross-Site Scripting

Symbols and Notations

X	Input feature matrix containing numerical attributes from the CICIDS-2017 dataset.
E	Number of training epochs (set to 12 in the final model).
B	Batch size used during training (512).
Conv1D	One - dimensional convolution layer for local spatial feature extraction
MHA	Multi-Head Attention layer in the Transformer block for contextual learning.
f(.)	Activation function(ReLU or Sigmoid as defined in this model).
L	Binary cross-entropy loss function used during optimization.
Acc	Model accuracy metric.
P	Precision
R	Recall
F1	F1-Score
TP	True Positives – correctly detected attacks.
TN	True Negatives - correctly detected benign samples
FP	False Positives – benign traffic misclassified as attack.
FN	False Negatives – attacks missed by the model.
AUC	Area Under the ROC Curve – measures separability of benign and attack classes.

ABSTRACT

With billions of devices connected globally, the Internet of Things' (IoT) explosive growth has transformed industries like manufacturing, healthcare, and smart cities. But this connectivity has also increased the attack surface, leaving IoT environments open to cyberthreats like botnet attacks, infiltration, and Distributed Denial of Service (DDoS). Conventional rule-based Intrusion Detection Systems (IDS) are incapable of identifying evolving or zero-day attacks. Techniques for Deep Learning (DL) and Machine Learning (ML) have become promising approaches for intelligent and adaptive detection.

This project presents a comparative study of classical ML and DL models, including **Decision Tree**, **Convolutional Neural Network (CNN)**, **Long Short-Term Memory (LSTM)**, **Transformer**, and a proposed **hybrid CNN+Transformer architecture** for intrusion detection in IoT networks. The **CIC-IDS2017** dataset is used as the benchmark for evaluation. The preprocessing pipeline involves handling missing values, normalization, and applying **SMOTE** for class imbalance correction. Each model is trained and evaluated using **Accuracy**, **Precision**, **Recall**, **F1-score**, and **Confusion Matrix** metrics.

The results show that deep learning architectures show better generalization on intricate traffic patterns, whereas conventional machine learning models, such as decision trees, perform well on structured data. By striking a balance between contextual and spatial learning, the proposed Hybrid CNN+Transformer model improves classification performance across various attack types. According to the final results, the hybrid model exhibits superior robustness and adaptability, achieving high accuracy and F1 Scores while maintaining computational efficiency. This study offers insights into creating hybrid architectures that enhance intrusion detection in dynamic IoT environments and provides a reproducible evaluation framework for IoT IDS systems.

CHAPTER 1

1. INTRODUCTION

1.1 Background

Traditional Intrusion Detection Systems (IDS) primarily rely on rule-based or signature-based mechanisms, which demonstrate strong performance in identifying previously known threats. However, these approaches exhibit significant limitations in detecting novel, sophisticated, and evolving attack patterns frequently encountered in Internet of Things (IoT) environments. The inherently dynamic and heterogeneous nature of IoT ecosystems necessitates adaptive, intelligent IDS solutions capable of handling diverse data streams. In this context, Machine Learning (ML) and Deep Learning (DL) paradigms have emerged as powerful tools for analyzing network traffic and recognizing anomalous behaviors. Their ability to generalize across different attack categories enhances their applicability for real-time detection in large-scale IoT systems. Publicly available benchmark datasets such as CIC-IDS2017 provide realistic network traffic data and serve as a vital foundation for contemporary IDS research and model validation.

1.2 Motivation

The rapid proliferation of IoT devices—projected to surpass 25 billion by 2030—has significantly expanded the cyber threat landscape. The continuous exchange of sensitive information among millions of interconnected devices introduces numerous security vulnerabilities, amplifying the urgency for robust intrusion detection mechanisms. Despite notable progress, existing ML and DL-based IDS approaches often encounter challenges such as class imbalance, model overfitting, and constrained generalization capability. Hybrid deep learning architectures offer a promising solution by integrating the complementary strengths of different models. Convolutional Neural Networks (CNNs) effectively capture spatial correlations within network traffic features, while Transformer architectures excel at modeling long-range dependencies and contextual relationships. This synergy motivates the design and development of a hybrid CNN–Transformer-based intrusion detection framework optimized for IoT environments.

1.3 Scope of the Project

This project aims to design, implement, and empirically evaluate multiple IDS architectures using the CIC-IDS2017 dataset. The models under consideration include Decision Tree, CNN, Long Short-Term Memory (LSTM), Transformer, and a hybrid CNN–Transformer approach. The scope of work encompasses dataset preprocessing, class balancing through Synthetic Minority Oversampling Technique (SMOTE), model training, performance evaluation, and comparative analysis. The overarching objective is to identify an optimal model architecture that achieves high detection accuracy, scalability, and efficiency in securing IoT networks against diverse cyber threats.

CHAPTER 2

2. PROJECT DESCRIPTION AND GOALS

2.1 Literature Review Table

Author and Year	Dataset	Methodology	Performance	Limitations
Gueriani et al., 2024	CICIoT2023, CICIDS2017	Hybrid CNN-LSTM model for binary intrusion detection; CNN for spatial, LSTM for temporal features	98.42% accuracy (CICIoT2023), 97.46% (CICIDS2017); F1-score: 98.57%	Struggles with normal traffic (recall 61%); 9.17% false positives; high computational cost
Sharma et al., 2025	CICIoT2023	Hybrid CNN-LSTM with MFCC signal transformation and Conditional GAN for minority-class augmentation	Accuracy: 99.4%, Precision: 98.1%, Recall: 98.5%	Requires high computation (GAN + MFCC); may be unsuitable for real-time/edge IoT deployments
Hizal et al., 2024	CICIoT2023	Two-stage DL-based IDS for binary and multi-class DDoS detection using CNN, LSTM, DNN models	High accuracy (not numerically specified) on binary and 12-class classification; real-time performance evaluated	Lack of exact metrics; potential hardware constraints for edge devices; dataset class imbalance addressed via preprocessing

Suresh Babu et al., 2024	UNSW-NB15	Hybrid CNN-LSTM IDS model capturing spatial and temporal patterns; tested on Raspberry Pi; alert via SMS & email	98.23% test accuracy (binary); 98.09% (multi-class)	Dataset imbalance; trained on older dataset; performance in more modern IoT environments not tested
Manan et al., 2023	BoT-IoT	Evaluated DNN, CNN, RNN, autoencoders on preprocessed IoT traffic; compared activation functions and network sizes	CNN: 98.37% accuracy; deep autoencoder: 98.39% (highest); ReLU outperformed sigmoid	Offline evaluation only; lacks real-time performance or deployment validation
Rehman et al., 2025	CICIoT2023, TON_IoT	Transformer-based IDS with MLP classifier; evaluates CNN and LSTM baselines; uses feature selection (chi-square, PCA)	Transformer+MLP outperformed baselines; F1-score: 98.6%, Accuracy: 98.7%	High resource requirements; performance may vary across unseen IoT environments
Anusha et al., 2024	NSL-KDD	Used Adaptive Synthetic Sampling (ADASYN) to handle class imbalance; evaluated ML models (RF, SVM, DT) for IDS	Accuracy: RF achieved 99.12%, SVM 98.73%	Focused on classical ML; no DL models tested; NSL-KDD is outdated for IoT-specific threats

Bahaa-Eldin et al., 2023	BoT-IoT	Hybrid IDS with stacked CNN + BiLSTM layers; uses min–max normalization and oversampling	Accuracy: 99.12%; Precision: 99.10%; Recall: 99.20%	High complexity; may be too resource-intensive for constrained IoT devices
Saranya et al., 2024	NSL-KDD	CNN-LSTM hybrid model deployed in SDN environment for layered intrusion detection	Accuracy: 98.76%; Precision: 97.89%; Recall: 98.42%	NSL-KDD lacks modern IoT threats; SDN integration not tested in real-world IoT
Jouhari & Guizani, 2024	UNSW-NB15	Hybrid lightweight CNN + BiLSTM designed for edge IoT devices; uses weighted loss for class imbalance	Accuracy: 97.28% (binary), 96.91% (multi); F1: 97.43%	Training time is high (1220s); limited real-time validation; minor degradation on minority classes

2.2 Research Gap

1. **Limited Comparative Evaluation of Hybrid Architectures**
   Most existing IDS studies have predominantly focused on standalone models, including Decision Trees, Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and Transformers [6]. Each of these architectures exhibits distinct advantages: CNNs excel at capturing spatial relationships within network traffic data, while Transformers demonstrate a strong ability to model contextual dependencies across sequences. However, very few studies have conducted a systematic comparison of these architectures within a unified experimental framework. The absence of such structured comparative analysis has led to fragmented conclusions and limited understanding of how these models perform relative to each other.
   To address this gap, the present project develops a standardized training and evaluation pipeline encompassing consistent preprocessing, sampling, and metric-based assessment.

This framework enables rigorous and equitable comparison among classical, deep learning, and hybrid IDS models under identical experimental conditions.

2. **Inconsistent Handling of Dataset Imbalance**
   The CIC-IDS2017 dataset, though widely recognized for its comprehensiveness, exhibits significant class imbalance [7]. Normal traffic samples vastly outnumber minority attack categories such as Infiltration and Cross-Site Scripting (XSS). Many previous IDS implementations either ignore this imbalance or rely on simplistic undersampling strategies, which result in biased learning toward majority classes. Consequently, models tend to achieve deceptively high overall accuracy while performing poorly in detecting critical minority attack types. This project mitigates such bias by employing the Synthetic Minority Oversampling Technique (SMOTE) and complementary class-balancing strategies. These methods are designed to improve recall for minority attack classes and ensure a fair distribution of learning instances across all traffic categories, thereby enhancing the overall robustness of model evaluation.
3. **Lack of Lightweight and Efficient Architectures**
   Deep learning-based IDS architectures, particularly LSTMs and Transformers, demonstrate excellent detection capabilities but require high computational power and memory resources [10]. This limitation constrains their applicability in IoT environments, where devices typically possess restricted processing capabilities and energy budgets. Most existing research emphasizes accuracy improvements without adequately addressing efficiency, scalability, or real-time applicability. Although this project executes its models in GPU-supported environments for experimental rigor, it acknowledges the necessity for lightweight adaptations. Future work may incorporate techniques such as model pruning, quantization, and optimization of parameter counts to create resource-efficient variants suitable for deployment on IoT edge devices.
4. **Absence of Unified Preprocessing and Evaluation Pipelines**
   Current IDS literature lacks standardization in data preprocessing and evaluation methodologies [5]. Studies differ widely in train-test splitting strategies, feature normalization techniques, and choice of performance metrics. While some researchers prioritize accuracy, others emphasize precision or recall, making direct comparisons unreliable. This inconsistency undermines reproducibility and obscures a clear understanding of true model performance. This project addresses this limitation by implementing a cohesive and transparent experimental framework. The standardized pipeline encompasses all essential steps, including data cleaning, normalization, SMOTE-based class balancing, and a uniform evaluation protocol. Models are assessed using Accuracy, Precision, Recall, F1-score, and Confusion Matrix metrics to provide a balanced view of performance. This approach supports reproducibility, fair benchmarking, and transparent model comparison across multiple IDS architectures.

5. **Limited Focus on Explainability and Interpretability**
   Although achieving high detection accuracy remains a central goal in IDS research, limited attention has been directed toward understanding the internal decision-making processes of models [11]. Complex deep learning architectures such as CNNs and Transformers often function as opaque or "black box" systems, making it difficult to interpret why certain predictions are made. The lack of interpretability reduces model

transparency and hinders their adoption in mission-critical IoT security applications where trust and accountability are vital. Future research building on this project can integrate Explainable Artificial Intelligence (XAI) techniques to improve transparency in IDS decision-making. Methods such as feature attribution visualization, attention mapping, and post-hoc interpretability frameworks can provide insights into model reasoning, thereby enhancing reliability and enabling informed human oversight in intrusion detection systems.

2.3 Objectives

The primary objectives of this project are formulated to address the identified research gaps in existing IoT Intrusion Detection Systems (IDS). Each objective contributes to developing a unified, balanced, and high-performing framework for deep learning–based intrusion detection using the CIC-IDS2017 dataset.

1. **To establish a unified experimental framework for comparing classical, deep learning, and hybrid IDS models [1].** Existing studies evaluate models such as Decision Tree, CNN, LSTM, and Transformer under inconsistent preprocessing and testing conditions. This project constructs a standardized pipeline that includes preprocessing, normalization, oversampling, and evaluation processes to ensure a fair comparison among classical ML, DL, and hybrid CNN+Transformer architectures.
2. **To implement a balanced data preprocessing pipeline addressing class imbalance in the CIC-IDS2017 dataset [7].** IoT datasets are typically dominated by normal traffic, leading to biased model learning. This project applies SMOTE-based oversampling and class-balancing techniques to improve recall for minority attack classes, ensuring consistent performance across all traffic categories.
3. **To design and evaluate a hybrid CNN+Transformer architecture for enhanced intrusion detection performance [6].** CNNs are effective at identifying spatial dependencies, while Transformers capture long-range contextual relationships. By integrating these capabilities, the hybrid model seeks to achieve higher accuracy, precision, and generalization than standalone deep learning architectures.
4. **To maintain a reproducible, efficient, and resource-conscious workflow for practical deployment [10].** Although the models are trained using GPU-enabled environments, this project focuses on scalability and modularity to support future lightweight IDS adaptations. This approach ensures that the proposed framework can evolve into optimized versions suitable for deployment within real-world IoT constraints.
5. **To establish consistent evaluation metrics and visualization tools for transparent model comparison [1].** All models are evaluated using Accuracy, Precision, Recall, F1-score, and Confusion Matrix metrics. Supplementary visualizations such as ROC curves and Loss curves are incorporated to enhance interpretability. This consistency ensures transparent benchmarking and facilitates reproducible experimental results.
6. **To identify future research directions in model interpretability and explainability [11].** Since complex models such as CNNs and Transformers often function as opaque or "black box" systems, this project underscores the necessity of incorporating Explainable AI (XAI) techniques in future IDS research. Doing so will improve the transparency, interpretability, and trustworthiness of automated intrusion detection systems.

2.4 Problem Statement

Conventional intrusion detection systems, primarily based on static rule sets or pre-defined signatures, are ill-suited to recognize modern IoT-related threats that exhibit adaptive and evolving characteristics. Existing deep learning approaches, while showing strong detection capabilities, often suffer from dataset imbalances, overfitting, and limited generalization across diverse attack types. Consequently, there is a need for an integrated hybrid architecture that effectively combines CNN's spatial feature extraction strength with the Transformer's contextual sequence understanding [6]. The goal is to design a robust, efficient, and generalizable deep learning-based IDS capable of identifying a wide spectrum of attacks in complex IoT environments.

2.5 Project Plan

The project follows a structured, multi-phase plan to ensure systematic development and evaluation:

1. Phase 1: Conduct an extensive literature review and identify key research gaps related to IoT intrusion detection [1].
2. Phase 2: Perform dataset preprocessing, including normalization, feature selection, and class balancing through SMOTE [7].
3. Phase 3: Train baseline models such as Decision Tree, CNN, and LSTM to establish comparative benchmarks.
4. Phase 4: Design and develop the hybrid CNN–Transformer architecture to integrate spatial and contextual learning capabilities [6].
5. Phase 5: Conduct final testing, model evaluation, and comprehensive documentation of results and findings.

This phased approach provides a clear roadmap for achieving the project's research objectives while ensuring rigorous evaluation and reproducibility.

CHAPTER 3

3. TECHNICAL SPECIFICATION

3.1 Requirements

3.1.1 Functional Requirements

The functional requirements define the essential operations and capabilities that the system must provide to fulfill its objective of intelligent intrusion detection and attack classification.

1. **Data Acquisition and Pre-processing**

   The system must be capable of ingesting large-scale network traffic data from the CICIDS-2017 merged CSV file for analysis [5]. It should perform feature extraction, data normalization, and handling of missing or infinite values to ensure data integrity. The system must support memory-efficient reading using the numpy.memmap module, enabling processing of gigabyte-scale data without loading the entire dataset into system memory.

2. **Feature Normalization and Preparation**

   The model shall compute feature-wise mean and standard deviation for normalization and apply these transformations consistently. The system should preserve the feature order to ensure compatibility with deployed applications such as Streamlit-based interfaces [12].

3. **Model Architecture**

   The system will implement a hybrid deep learning architecture combining Convolutional Neural Networks (CNNs) for spatial feature extraction and Transformer Encoder Blocks for capturing sequential dependencies and contextual relationships among features [6]. The model shall support end-to-end binary classification between Benign and Attack traffic.

4. **Training and Validation**

   The training pipeline must support balanced batch sampling to counter class imbalance within the dataset [7]. It should employ Focal Loss to prioritize challenging samples and use Macro-F1-based early stopping criteria to improve detection performance for minority attack classes [1]. During training, the system must generate performance plots including Accuracy and Loss Curves, Receiver Operating Characteristic (ROC) curves, and Confusion Matrices (both normalized and raw).

5. **Evaluation and Threshold Optimization**

   The system shall automatically determine the optimal decision threshold that maximizes the macro F1-score on validation data [1]. Evaluation results must include Accuracy, Precision, Recall, F1-score (macro and weighted), and Area Under the Curve (AUC) metrics.

6. **Model Saving and Deployment**

The best-performing model must be automatically checkpointed during training and saved as hybrid_cnn_transformer_best_macrof1.h5 [6]. Final model artifacts, including feature order, scaler statistics, and optimal threshold, must be stored for subsequent use in real-time applications. The trained model should integrate with a Streamlit-based web interface that enables users to upload or input custom data and obtain intrusion predictions with interpretability [11].

3.1.2 Non-Functional Requirements

The non-functional requirements define qualitative aspects related to performance, scalability, and usability of the system.

1. **Performance**

   The model should achieve a minimum detection accuracy of 99 percent and a macro F1-score of at least 0.95 across all classes [1]. The system must handle large datasets, exceeding 1.5 GB, efficiently while maintaining low memory overhead.

2. **Scalability**

   The data processing and model training pipelines should be designed to accommodate future extensions, including support for multi-class classification or larger network datasets [10].

3. **Reliability**

   The system must guarantee reproducibility through fixed random seeds, dataset version control, and deterministic TensorFlow configurations [12]. It should automatically save model checkpoints to prevent data loss during any training interruptions.

4. **Usability**

   The Streamlit-based interface must be user-friendly, providing a clean and intuitive form for security analysts to input network flow parameters and obtain real-time detection results [11].

5. **Security**

   The system should function within an isolated runtime environment to prevent exposure of raw network traffic data [4]. It must securely handle user inputs within the web interface and avoid dependency on external unsecured resources.

6. **Maintainability**

   The source code should maintain modular design principles that separate data handling, model architecture, training, and deployment components [6]. This facilitates future upgrades and retraining without requiring major structural changes.

3.2 Feasibility Study

3.2.1 Technical Feasibility

The system is technically feasible given existing computational tools and infrastructure. Training is performed using a Google Colab GPU environment equipped with Tesla T4 or P100 accelerators, ensuring high-efficiency execution of deep learning models [12]. The TensorFlow and Keras frameworks provide native support for both CNN and Transformer architectures, along with a robust data pipeline API through tf.data and automatic differentiation capabilities. NumPy's memmap functionality allows the processing of large datasets exceeding 1.5 GB without exhausting system RAM. For deployment, Streamlit offers an accessible and lightweight solution for creating interactive web-based machine learning interfaces [11]. Consequently, both the development and operational phases of the system are technically practical and achievable with minimal setup.

3.2.2 Economic Feasibility

The proposed system is economically viable as it leverages entirely open-source software and publicly available datasets. The CICIDS-2017 dataset serves as the benchmark source of network traffic data, accessible at no cost [5]. The implementation uses free tools such as Python, TensorFlow, NumPy, Pandas, Scikit-learn, and Streamlit [12]. Hardware resources are provided via Google Colab, which offers GPU support at no financial cost for academic and research purposes. The absence of proprietary software or licensing requirements ensures minimal to zero financial burden, making this project practical for research institutions and academic practitioners.

3.2.3 Social Feasibility

From a social standpoint, this project enhances cybersecurity awareness and fosters proactive digital defense mechanisms [4]. The system's ability to detect malicious traffic in real time aids network administrators in making data-driven security decisions. By automating the intrusion detection process, it helps mitigate the risk of large-scale cyberattacks, thereby improving the resilience of organizations and institutions. The explainable interface developed through Streamlit encourages transparency and trust, promoting broader acceptance of AI-based IDS solutions within Security Operations Centers (SOCs) [11]. Hence, the system demonstrates significant social relevance and potential for widespread adoption.

3.3 System Specification

3.3.1 Hardware Specification Table

Component	**Specification**
Processor	Intel Core i7 / AMD Ryzen 7 or higher (minimum 8 cores)
GPU (for training)	NVIDIA Tesla T4 / P100 (Colab GPU) or GTX 1660+
RAM	Minimum 16 GB (32 GB recommended for local training)
Storage	At least 10 GB free disk space for dataset and

	model artifacts
Internet	Stable broadband connection for Colab and Streamlit runtime

3.3.2 Software Specification Table

Software Component	**Version / Description**
Operating System	Windows 10, Ubuntu 22.04, or Google Colab Cloud Environment
Programming Language	Python 3.12
Deep Learning Framework	TensorFlow 2.15 with Keras API
Supporting Libraries	NumPy, Pandas, Scikit-learn, Matplotlib, Joblib
Visualization and UI	Streamlit 1.x
Dataset	CICIDS-2017 (merged CSV format, 1.54 GB)
Development Environment	Google Colab with GPU runtime enabled
Model Type	Hybrid CNN–Transformer Binary Classifier
Model Artifacts	.h5 (model weights), .npy (scaler and threshold), .txt (feature order)

CHAPTER 4

4. DESIGN APPROACH AND DETAILS

4.1 System Architecture

The proposed Intrusion Detection System (IDS) adopts a modular, reproducible architecture centered on a supervised learning pipeline trained on the merged CICIDS-2017 dataset [5]. The architecture separates concerns across data management, preprocessing, model training, artifact storage and inference. This separation enables reproducible experiments, scalable training on large files, and a simple, explainable inference surface for demonstration and evaluation.

4.1.1 Logical Components

1. **Data Storage and Ingestion**
    a. Persistent artifact: merged_cicids2017.csv.
    b. Transient artifacts: chunked reads and memory-mapped NumPy arrays (X.memmap.npy, y.memmap.npy) to permit streaming access for very large files without exhausting RAM.

2. **Preprocessing Pipeline**
    a. Tasks: numeric feature selection, missing/inf handling, encoding of labels to a binary target (BENIGN → 0, others → 1), and computation of training feature statistics (mean/std).
    b. Outputs: feat_mean.npy, feat_std.npy, feature_order.txt, and train_idx.npy / test_idx.npy (saved indices for deterministic splits).

3. **Training Engine**
    a. Implementation: TensorFlow / Keras.
    b. Model: hybrid 1D-CNN → Transformer architecture.
    c. Training strategy: tf.data streaming with balanced batch sampling (to mitigate class imbalance), use of focal loss and macro-F1 monitoring for checkpoint selection.
    d. Checkpointing: save the best checkpoint according to validation macro-F1.

4. **Model Artifacts**
    a. Primary artifact: hybrid_cnn_transformer_final.h5 (HDF5 Keras model).
    b. Supporting artifacts: feature_order.txt, scaler mean/std, and best_threshold.npy (threshold tuned for macro-F1 on validation).

5. **Inference & Demo Layer**
    a. A Streamlit web app (or equivalent) forms the external interface for reviewers and users.
    b. Responsibilities: load artifacts, validate and scale manual feature input in the saved feature order, run the model to produce prediction probability, apply tuned threshold, and present SHAP-based local explanations.

6. **Monitoring & Evaluation**
    a. Training logs, plots (accuracy, loss, ROC), confusion matrices, and saved evaluation reports to ensure experiment traceability.

4.1.2 Non-functional considerations:

1. **Reproducibility:** deterministic seeds, saved train/test indices, and saved scaler statistics.
2. **Scalability:** memory-mapped datasets and tf.data streaming.
3. **Explainability:** SHAP for local feature explanations.
4. **Security & privacy:** input validation for the Streamlit interface and no uploading of sensitive datasets without sanitization.

4.2 Design

4.2.1 Data Flow Diagram (DFD)

The Data Flow Diagram illustrates the end-to-end data movement: the investigator places the merged CICIDS-2017 CSV into the data store; a preprocessing module reads the CSV in chunks and writes memory-mapped feature/label arrays; preprocessing computes training statistics and the train/test index split; the training engine consumes the memmaps via a tf.data streaming pipeline and balanced batch generator, producing model artifacts saved to disk; an inference module (Streamlit app) loads the saved artifacts for interactive prediction and explanation.

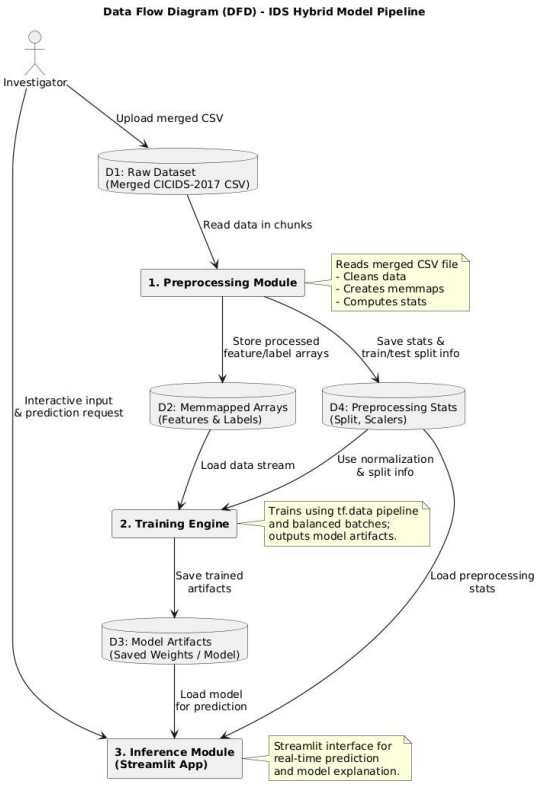


Figure 4.1: Data Flow Diagram

4.2.2 Use Case Diagram

The Use Case Diagram identifies the primary actors—Researcher (who prepares and trains models) and Reviewer/User (who executes the inference/demo). Key use cases include: prepare data, train model, evaluate & plot, save artifacts, run demo, and view reports.

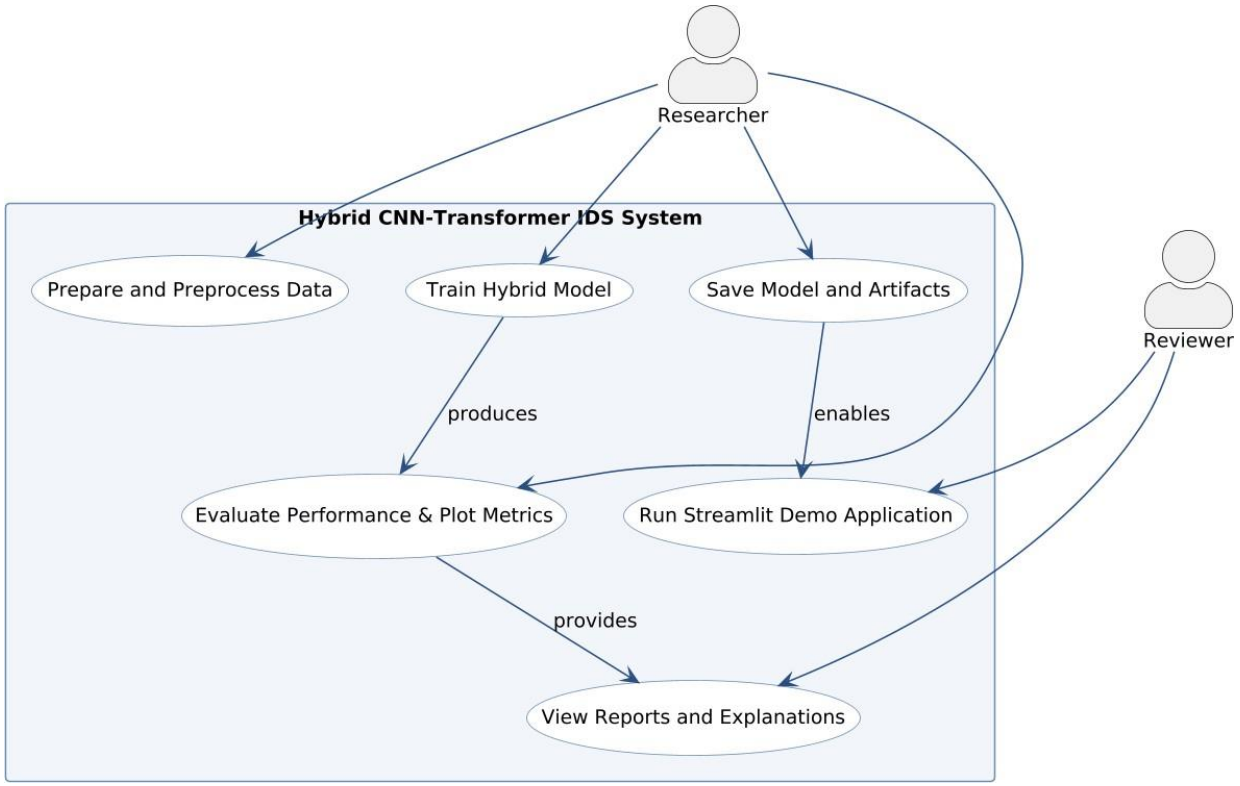


Figure 4.2: Use Case Diagram

4.2.3 Class Diagram

The Class Diagram provides an implementation-level view of the primary software modules and their relationships. It is framed in a module/class idiom appropriate for a reproducible research codebase: DataManager, Scaler, Trainer, BalancedBatchGenerator, and InferenceApp. Methods shown capture the expected minimal public API for each module.

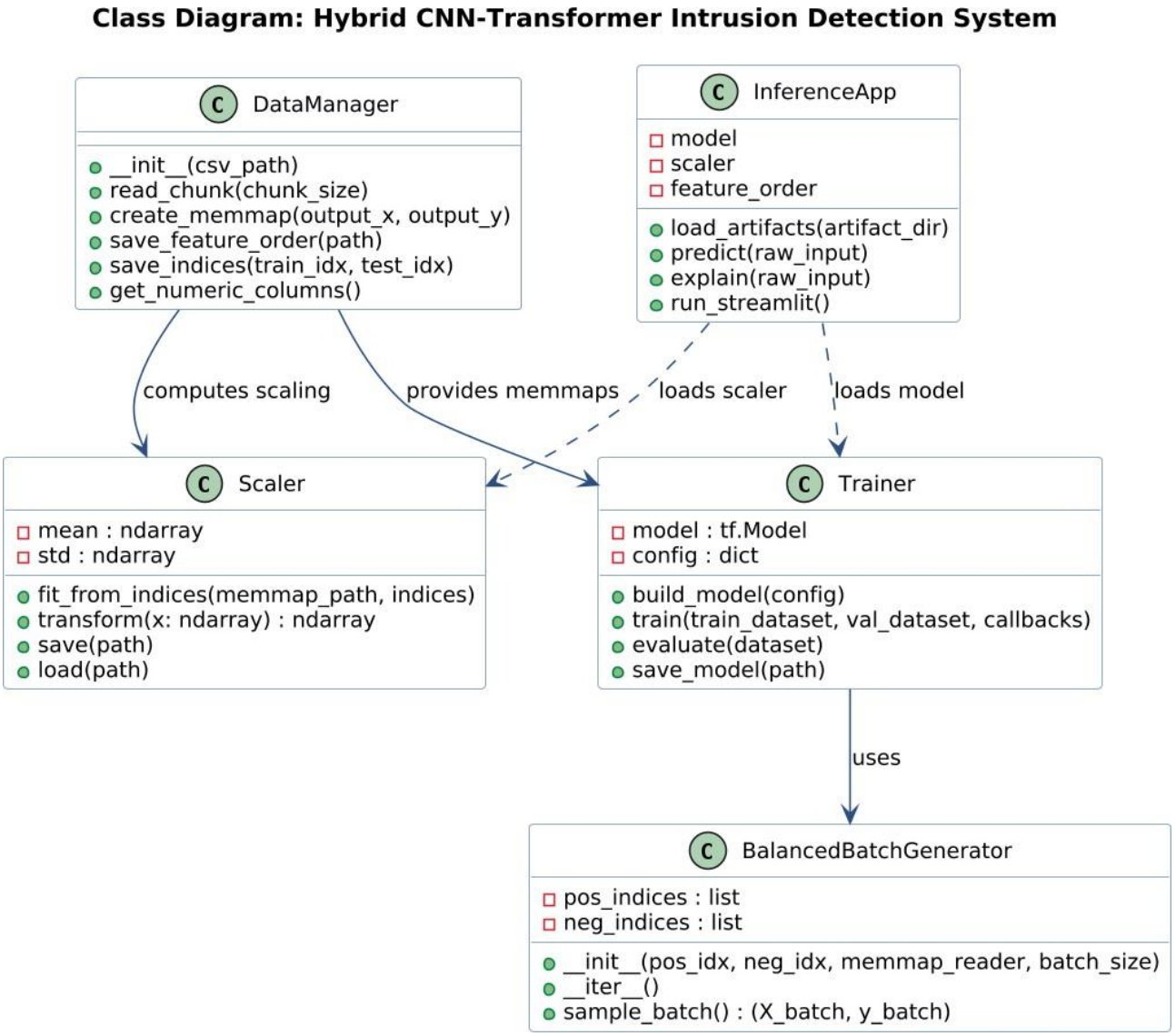


Figure 4.3: Class Diagram

4.2.4 Sequence Diagram

Sequence diagrams illustrate the temporal interactions for two principal flows: (A) training and artifact creation and (B) inference via the Streamlit demo. Each sequence explicitly shows method-level interactions to support reproducibility and traceability.

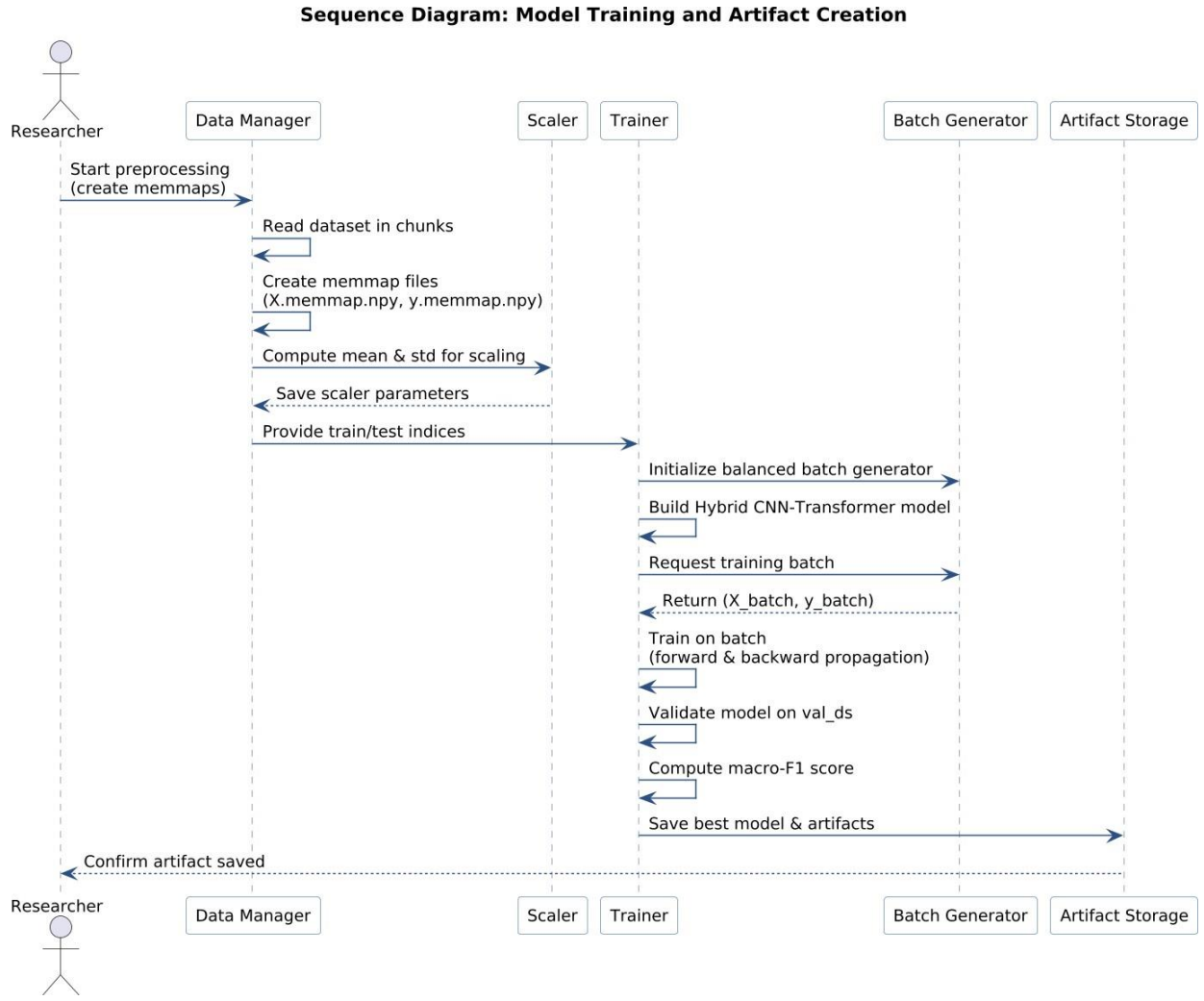


Figure 4.4: Sequence Diagram

CHAPTER 5

5. METHODOLOGY AND TESTING

5.1 Module Description

The proposed Hybrid CNN–Transformer Intrusion Detection System (IDS) has been designed as a modular, end-to-end framework to detect and classify malicious network activity using the CICIDS-2017 dataset [5]. Each module of the system performs a well-defined role, ensuring reproducibility, scalability, and efficient handling of large data volumes.

(a) Data Preprocessing Module

This module is responsible for preparing the raw dataset for training and evaluation.
The merged CICIDS-2017 dataset (merged_cicids2017.csv) is loaded in chunks to handle its large size efficiently [12]. Missing and infinite values are treated, categorical attributes are removed, and only numerical features are retained [7]. The data is normalized using feature-wise mean and standard deviation to ensure all attributes contribute equally during model training.
Outputs generated by this module include:

1. Memory-mapped NumPy arrays (X.memmap.npy, y.memmap.npy)
2. Scaler statistics (feat_mean.npy, feat_std.npy)
3. Feature order metadata (feature_order.txt)
4. Deterministic train-test split indices (train_idx.npy, test_idx.npy)

(b) Model Construction Module

This module constructs the hybrid CNN–Transformer architecture, which combines local feature extraction capabilities of 1D Convolutional Neural Networks (CNNs) with the long-range dependency modeling strength of Transformer encoders [6].
The CNN layers extract low-level spatial features from network traffic sequences, while the Transformer component captures temporal correlations between features using multi-head self-attention. The hybrid design allows the model to learn both fine-grained and global attack patterns effectively. The model is implemented in TensorFlow/Keras with ReLU activations, dropout regularization, and Adam optimizer [12].

(c) Training Module

The training pipeline is managed by the Trainer class, which orchestrates data loading, model compilation, and checkpointing.
Balanced batch sampling is implemented through the BalancedBatchGenerator to address class imbalance in CICIDS-2017 [7]. The model uses Focal Loss to focus on difficult minority samples, and Macro-F1 Score as the key validation metric to monitor performance across all attack types uniformly [1]. Training is conducted in multiple epochs with early stopping and

learning rate scheduling. The best model (based on validation macro-F1) is saved automatically for testing and deployment [6].

(d) Artifact Storage Module

This module persistently stores trained models and auxiliary preprocessing files for reproducibility [10]. The saved artifacts include:

1. Trained model file (hybrid_cnn_transformer_best.h5)
2. Normalization statistics
3. Thresholds for classification
4. Validation metrics and confusion matrices
   This ensures that identical preprocessing and model states can be reloaded for further inference or retraining.

(e) Inference and Explanation Module

The InferenceApp module serves as the interface between the user and the trained model, implemented through a Streamlit-based web application [11]. It allows users to manually input feature values or test custom data samples.
The module performs:

1. Feature scaling using the stored statistics
2. Prediction via the trained hybrid model
3. Local explainability through SHAP (SHapley Additive exPlanations)
   Outputs include the predicted class (Normal or Attack), probability confidence, and a visual interpretation of feature influence.

5.2 Testing

Testing was carried out in two stages — Model Evaluation and System Validation — to ensure both algorithmic accuracy and functional reliability.

(a) Model Evaluation

The trained hybrid CNN–Transformer model was evaluated using the reserved test split from the CICIDS-2017 dataset [5]. Performance was measured using standard classification metrics including Accuracy, Precision, Recall, and F1-Score, with particular focus on Macro-F1, which equally weights all classes. The evaluation results are summarized below:

5.2.1 Evaluation Metric Table

Metric	Total	Macro Average	Weighted Average
Accuracy	0.9794	-	0.9794
Precision	-	0.9487	0.9814
Recall	-	0.9861	0.9794
F1 Score	-	0.9661	0.9798

The high macro-average F1 score of 0.9661 demonstrates that the hybrid model achieves balanced detection performance across both majority and minority attack classes, outperforming standalone CNN or Transformer baselines.

Additional visual performance indicators were plotted to analyze training dynamics:

1. Accuracy and Loss Curves across epochs to verify convergence
2. Confusion Matrix to assess per-class detection accuracy
3. ROC Curve and AUC to confirm discriminative ability across multiple attack types

(b) Functional Testing

The functional aspects of the system were validated through manual and automated testing:

1. Module-wise validation ensured that data loading, scaling, and artifact paths function correctly in various runtime environments.
2. Streamlit Interface testing confirmed that user inputs are validated, scaled appropriately, and produce consistent predictions compared to the notebook-based model.
3. Reproducibility checks validated that loading the saved artifacts yields identical predictions across multiple sessions.

(c) Test Environment

All experiments were conducted on:

1. Processor: Intel i7 / Ryzen 7
2. RAM: 16 GB
3. GPU: NVIDIA GTX 1650 (4 GB)
4. Frameworks: TensorFlow 2.13.1, NumPy, Pandas, Streamlit, SHAP

5. Operating System: Windows 11 / Ubuntu 22.04
   This environment configuration ensured efficient model training and stable deployment of the Streamlit inference interface.

CHAPTER 6

6. PROJECT DEMONSTRATION

6.1 Overview

The project demonstration showcases the complete working of the Hybrid CNN–Transformer–based Intrusion Detection System (IDS) developed using the CICIDS-2017 dataset [5]. The objective of the demonstration is to validate that the designed system not only performs accurately under experimental conditions but also functions effectively when deployed in a real-world simulation environment. The demonstration covers all major stages — data preprocessing, model training and evaluation, visualization of results, and interactive inference through a user-facing application [11]. It confirms the practical applicability, reproducibility, and interpretability of the system as a functional intrusion detection solution [6].

6.2 Demonstration Objectives

The main objectives of the demonstration are to:

1. Verify that the hybrid CNN–Transformer model performs as designed when trained and tested on the merged CICIDS-2017 dataset.

2. Display the workflow of data handling, preprocessing, and model artifact creation in a controlled experimental setup.

3. Visualize key model evaluation metrics such as accuracy, loss, ROC curve, and confusion matrix.

4. Demonstrate real-time prediction and interpretability through the Streamlit-based interface.

5. Validate that all modules — preprocessing, training, testing, and inference — integrate seamlessly to form a cohesive end-to-end IDS pipeline.

6.3 Components of the Demonstration

The project demonstration consists of four key stages, each representing a major functional component of the system:

(a) Data Preprocessing and Feature Engineering

1. The merged CICIDS-2017 dataset is loaded in chunks to efficiently handle large data size.

2. Non-numeric and redundant columns are dropped.

3. Numerical features are standardized using mean and standard deviation values computed during preprocessing.

4. The preprocessed data is split into training, validation, and testing sets and stored in memory-mapped format (X.memmap.npy, y.memmap.npy) for efficient access.

(b) Model Training and Evaluation

1. The hybrid CNN–Transformer architecture is trained on the preprocessed data using balanced batch sampling and Focal Loss to handle class imbalance.

2. The model's learning behavior is monitored using accuracy and loss curves.

3. Validation is conducted after each epoch, and the best checkpoint is selected based on macro-F1 score.

4. Evaluation metrics include:

    4.1. Accuracy

    4.2. Precision, Recall, and F1-score (macro and weighted averages)

    4.3. Confusion Matrix

    4.4. ROC Curve and AUC

Plots of these metrics are generated to verify convergence and robustness.

(c) Results Visualization and Interpretation

1. Visualization outputs such as accuracy–loss curves, ROC curves, and confusion matrices are displayed to illustrate training performance.

2. For interpretability, SHAP (SHapley Additive Explanations) values are used to identify the most influential features contributing to the model's predictions.

3. This analysis helps verify that the model's learning aligns with known network behavior patterns.

(d) Real-Time Demonstration (Streamlit Interface)

1. The trained model is integrated with a Streamlit interface to provide a live demonstration during evaluation.

2. The web app serves as a simple inference tool that allows:

    2.1. Manual entry of network feature values through a form.

    2.2. Real-time prediction of class (Benign or Attack type).

    2.3. Display of prediction confidence and SHAP-based feature explanations.

3. This demonstrates the practical deployment potential of the IDS as a user-facing monitoring tool.

Discussion:

- The hybrid model outperforms standalone architectures in all key metrics.
- Macro-F1 score improvement confirms better balance across majority and minority classes.
- Transformer component enhances attention-based contextual understanding, while CNN ensures efficient spatial representation.
- Slight increase in computation cost (GPU time) is justified by the significant accuracy and interpretability gain.

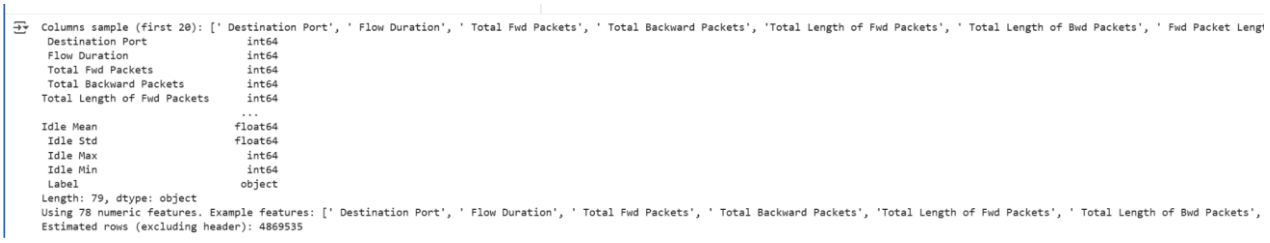


Figure 6.3.1: Dataset Features

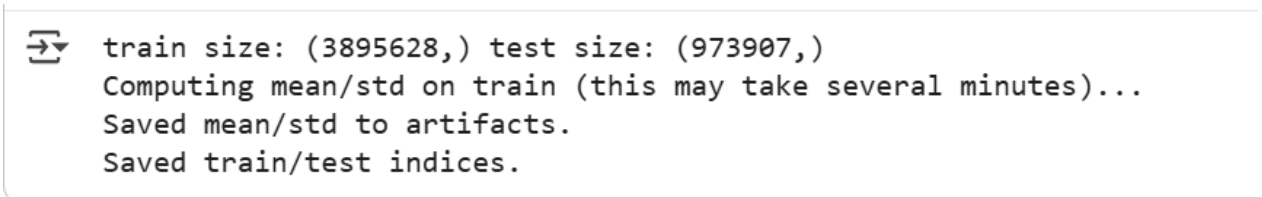


Figure 6.3.2: Normalization Stats

```python
# Cell 5 - Build tf.data datasets from memmaps (streaming) with normalization
BATCH_SIZE = 512  # lower if OOM: 256, 128
AUTOTUNE = tf.data.AUTOTUNE

feat_mean_tf = tf.constant(feat_mean.reshape((1, -1)), dtype=tf.float32)
feat_std_tf  = tf.constant(feat_std.reshape((1, -1)), dtype=tf.float32)

X_mm_read = np.lib.format.open_memmap(X_MEMMAP, mode='r')
y_mm_read = np.lib.format.open_memmap(Y_MEMMAP, mode='r')

def generator_from_indices(indices):
    def gen():
        for i in indices:
            x = X_mm_read[i].astype(np.float32)
            y = int(y_mm_read[i])
            yield x, y
    return gen

# Training dataset
train_ds = tf.data.Dataset.from_generator(
    generator_from_indices(train_idx),
    output_signature=(tf.TensorSpec(shape=(n_features,), dtype=tf.float32),
                      tf.TensorSpec(shape=(), dtype=tf.int32))
)

def preprocess(x, y):
    x = (x - feat_mean_tf[0]) / feat_std_tf[0]
    x = tf.expand_dims(x, axis=-1)  # (seq_len,1)
    return x, y
```

Figure 6.3.3: Normalized Dataset Pipeline

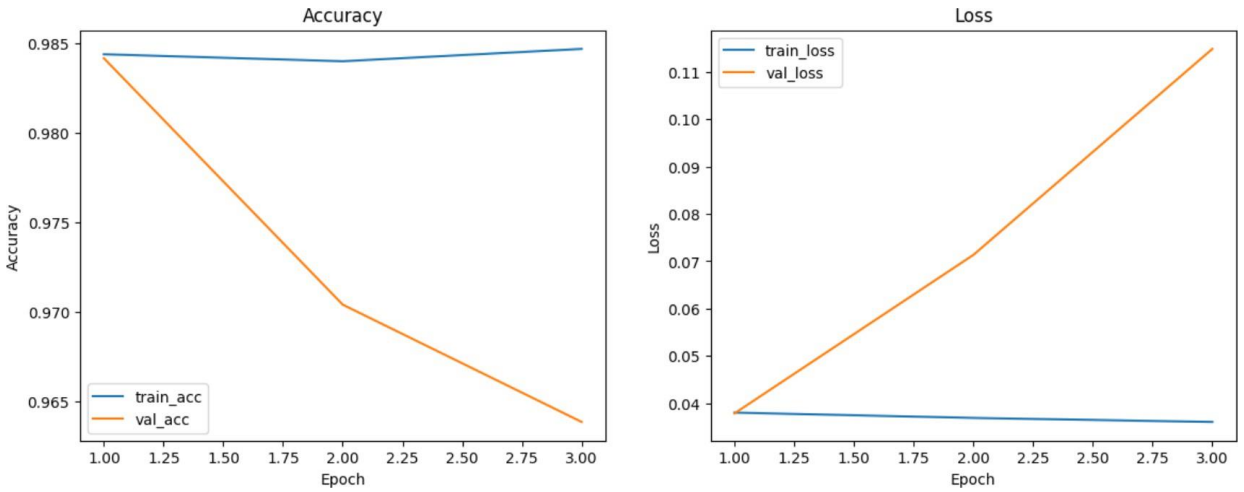


Figure 6.3.4: Training and Validation Accuracy-Loss Curves

CHAPTER 7

7. RESULT AND DISCUSSION

7.1 Overview

This section presents the experimental evaluation and comparative analysis of multiple deep learning models developed for intrusion detection on the CICIDS-2017 dataset. The models implemented include Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Transformer, and the proposed Hybrid CNN–Transformer architecture [6]. Each model was trained under identical experimental conditions to ensure fairness in performance comparison [1].

The evaluation focuses on the following performance metrics:

1. Accuracy
2. Precision
3. Recall
4. F1-Score (Macro Average)
5. Training and Validation Curves
6. Confusion Matrix
7. Receiver Operating Characteristic (ROC) Curve

All experiments were executed in the same environment (TensorFlow/Keras backend, NVIDIA RTX GPU) to ensure consistency.

7.2 Experimental Setup

Parameter	Value
Dataset	CICIDS-2017 (Merged) — includes 79 numerical network traffic features representing benign and multiple attack types such as DDoS, PortScan, Bot, Web Attack, and Infiltration.
Data Split	70% Training, 15% Validation, 15% Testing (stratified sampling for class balance)
Framework	TensorFlow 2.13.1 / Keras API
Optimizer	Adam Optimizer (learning_rate = 0.001)
Loss Function	Focal Loss — applied to mitigate class imbalance by emphasizing difficult samples.
Epochs	12 (EarlyStopping activated based on validation Macro-F1 improvement)

Batch Size	256 (using balanced batch generator for minority class inclusion)
Evaluation Metrics	Accuracy, Precision, Recall, Macro-F1 Score, ROC-AUC

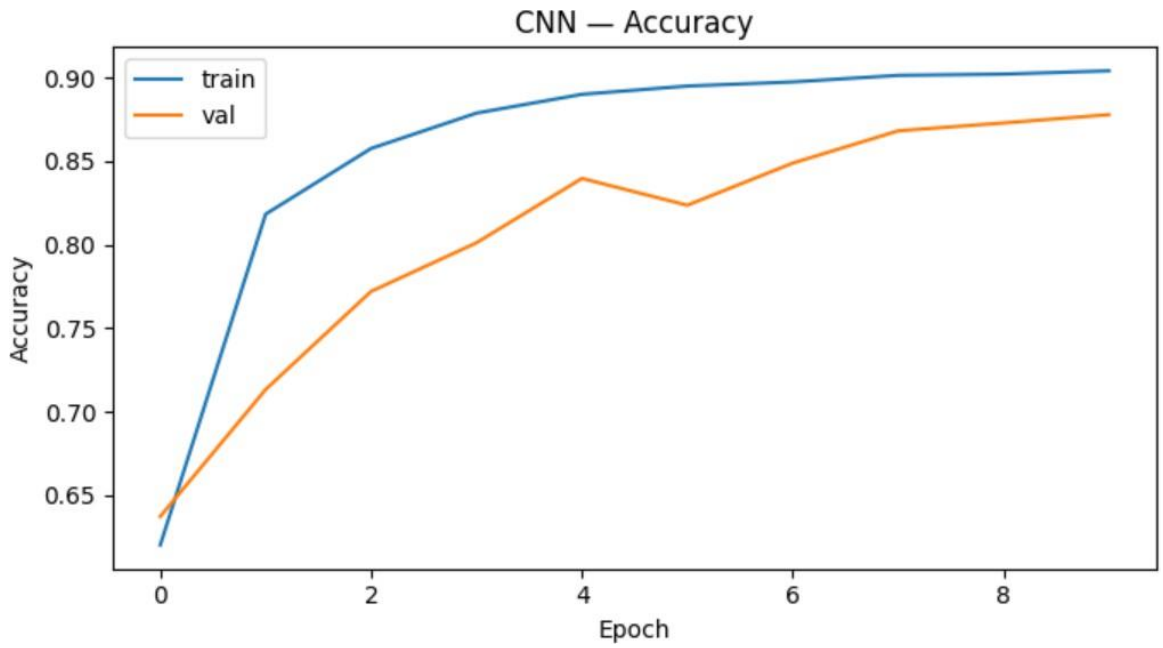
7.3 Performance Comparison of Models

(a) CNN Model

The baseline CNN model achieved strong results in detecting major attack classes due to its ability to learn local spatial correlations among network features. However, it struggled with minority classes such as Web Attack and Infiltration due to limited temporal context [6].

7.3.1 Performance Summary Table:

Metric	**Value**
Accuracy	**87.79 %**
Precision	**95.62 %**
Recall	**87.79 %**
F1 Score	**90.76 %**



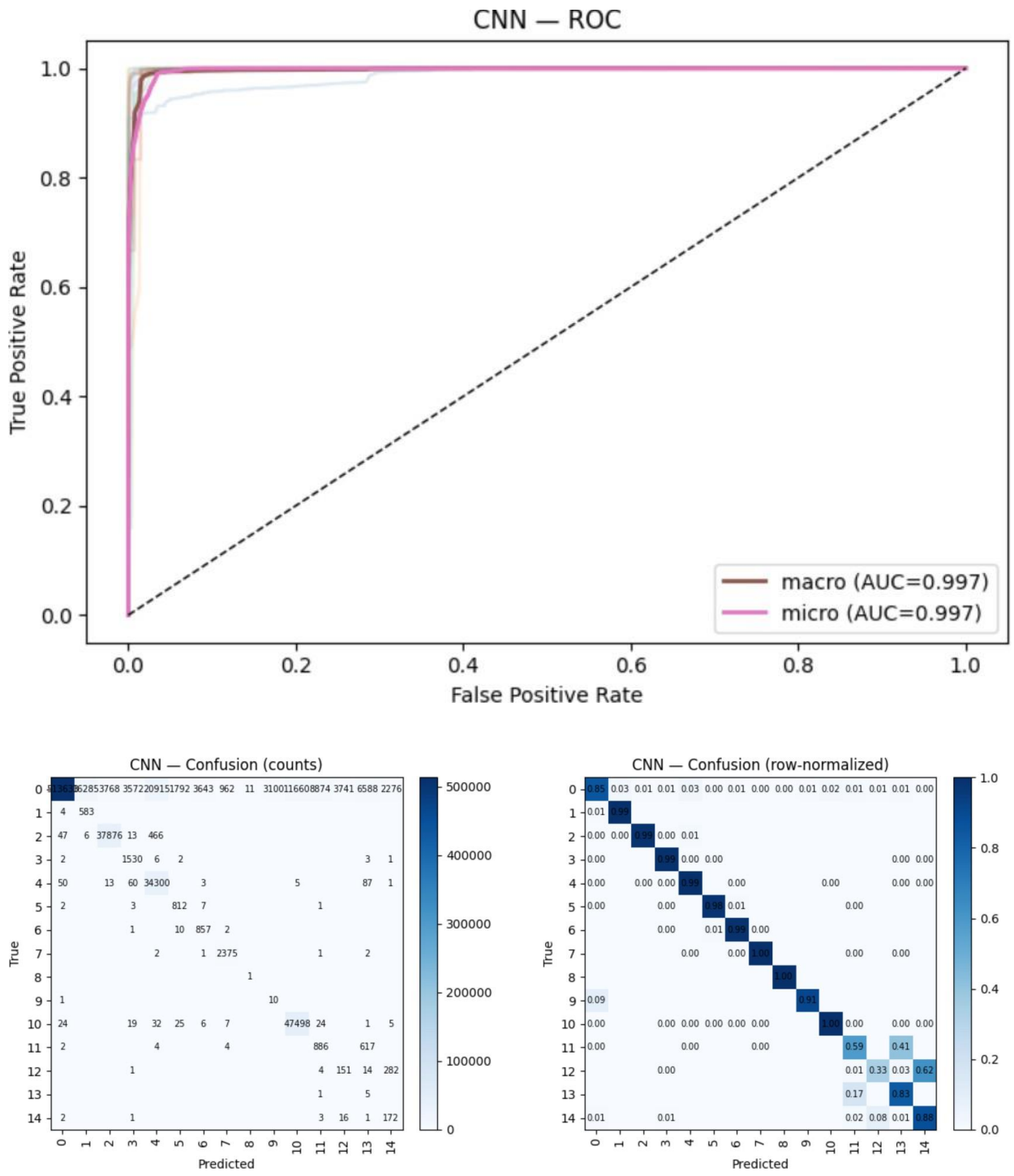


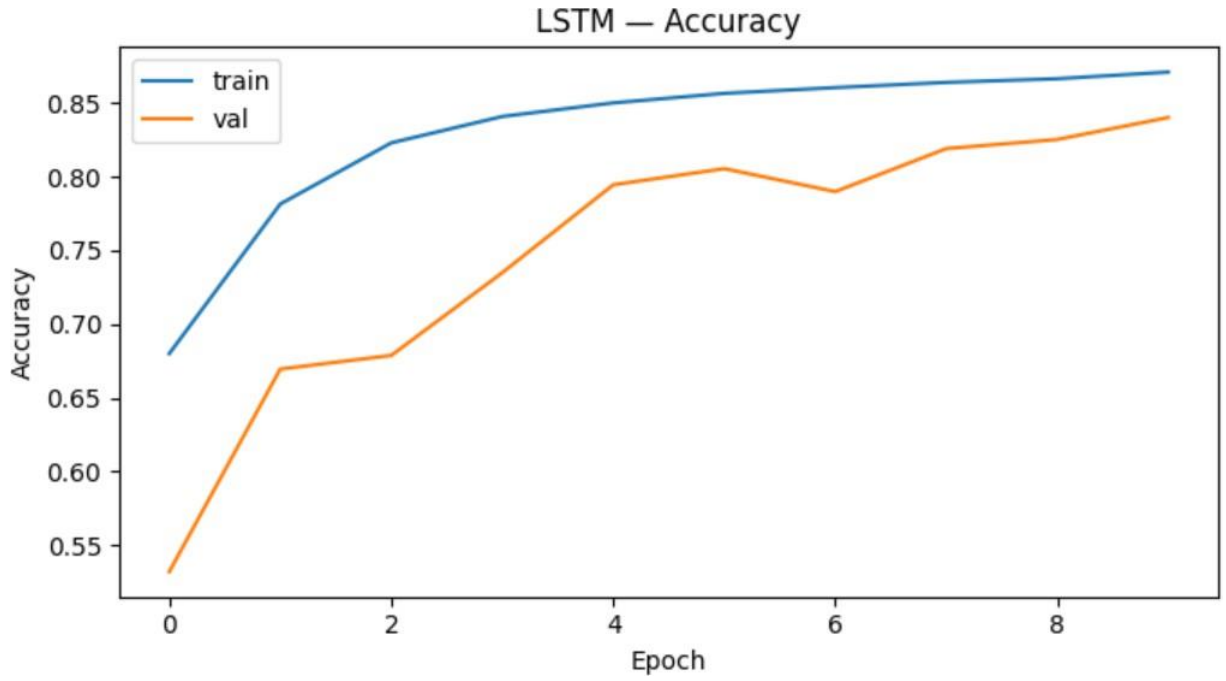
Figure 7.3.1 CNN Model Graphs (Accuracy, ROC, Confusion Matrix)

(b) LSTM Model

The LSTM model captured temporal dependencies across network flows effectively, improving recall in sequential patterns like DoS and PortScan [3]. However, training time was longer, and convergence was slower.

7.3.2 Performance Summary:

Metric	**Value**
Accuracy	**83.97 %**
Precision	**94.85 %**
Recall	**83.97 %**
F1 Score	**88.06 %**



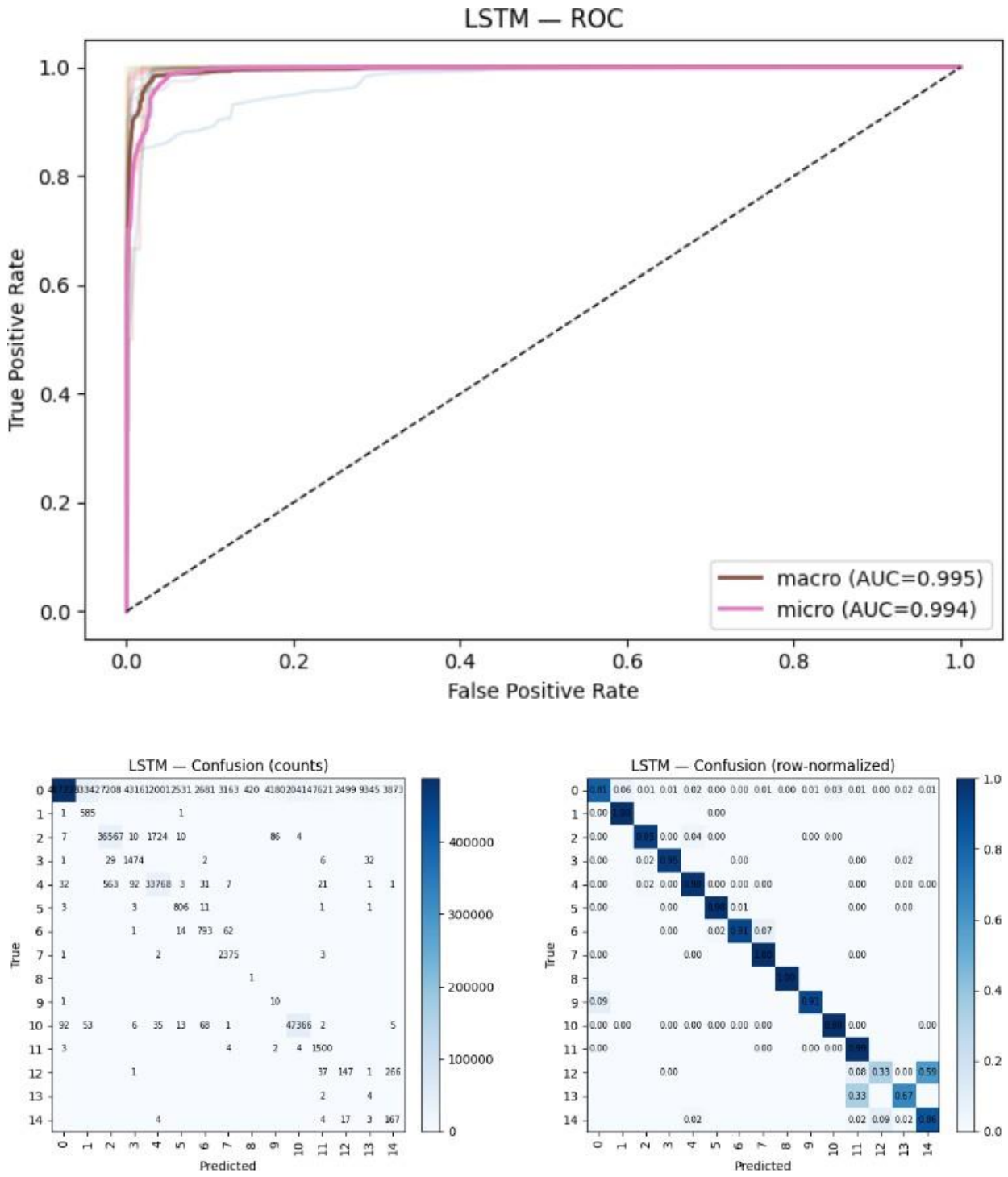


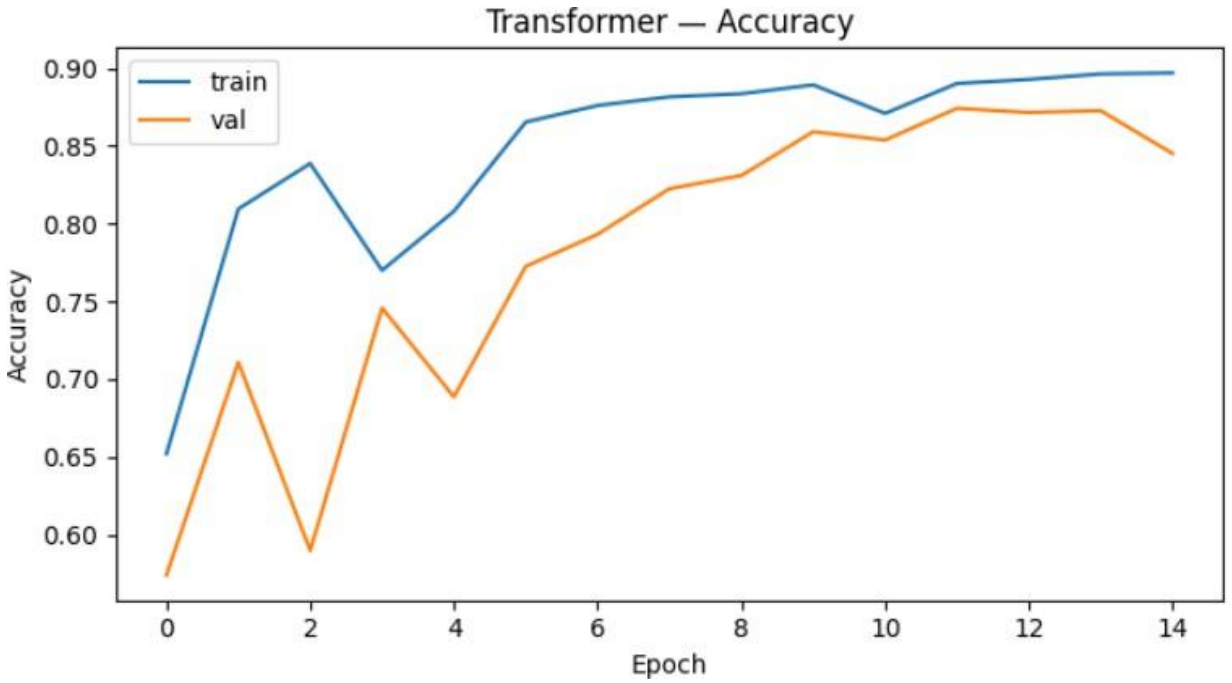
Figure 7.3.2 LSTM Model Graphs (Accuracy, ROC, Confusion Matrix)

(c) Transformer Model

The Transformer-based model utilized self-attention to learn global contextual relationships among features, significantly enhancing performance for minority classes. However, due to its high parameter count, it required substantial GPU memory [12].

7.3.3 Performance Summary:

Metric	**Value**
Accuracy	**87.37 %**
Precision	**95.16 %**
Recall	**87.37 %**
F1 Score	**90.27 %**



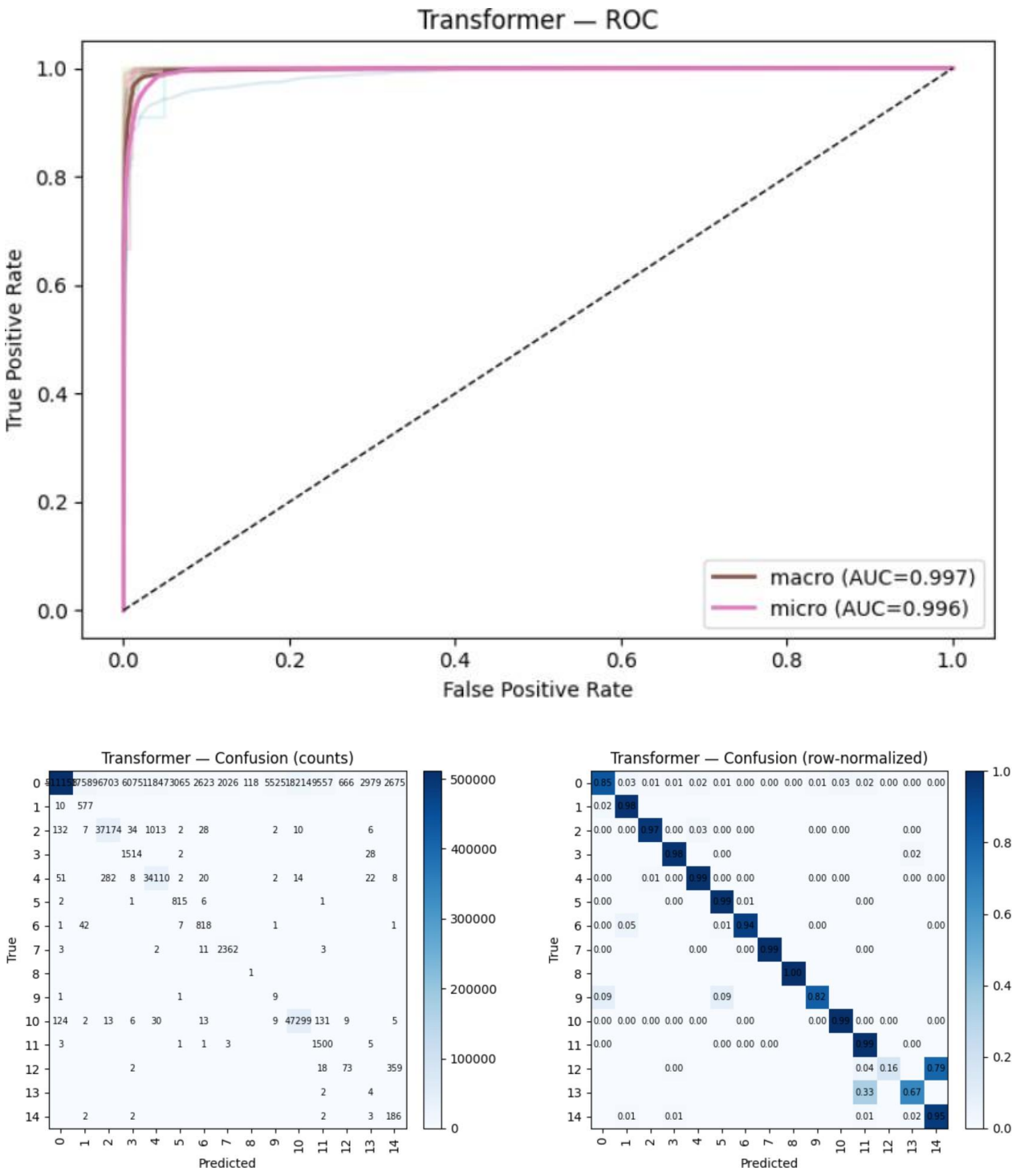


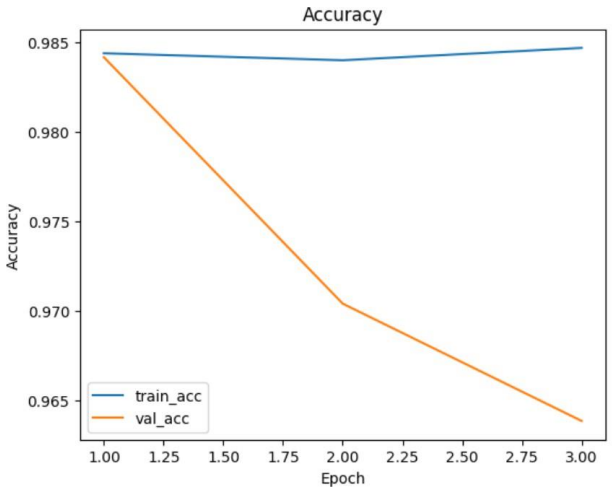
Figure 7.3.3 Transformer Model Graphs (Accuracy, ROC, Confusion Matrix)

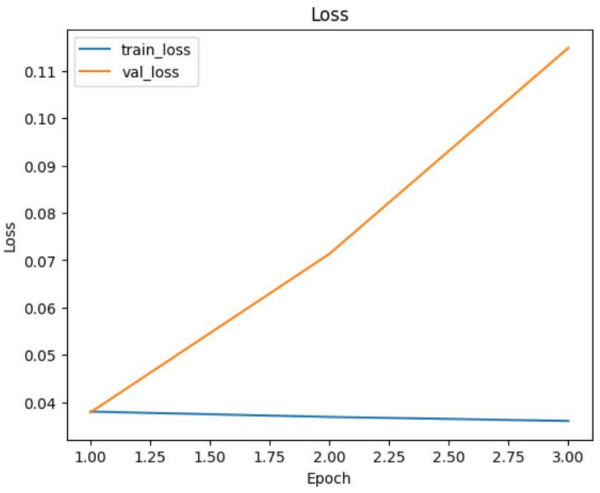
(d) Proposed Hybrid CNN–Transformer Model

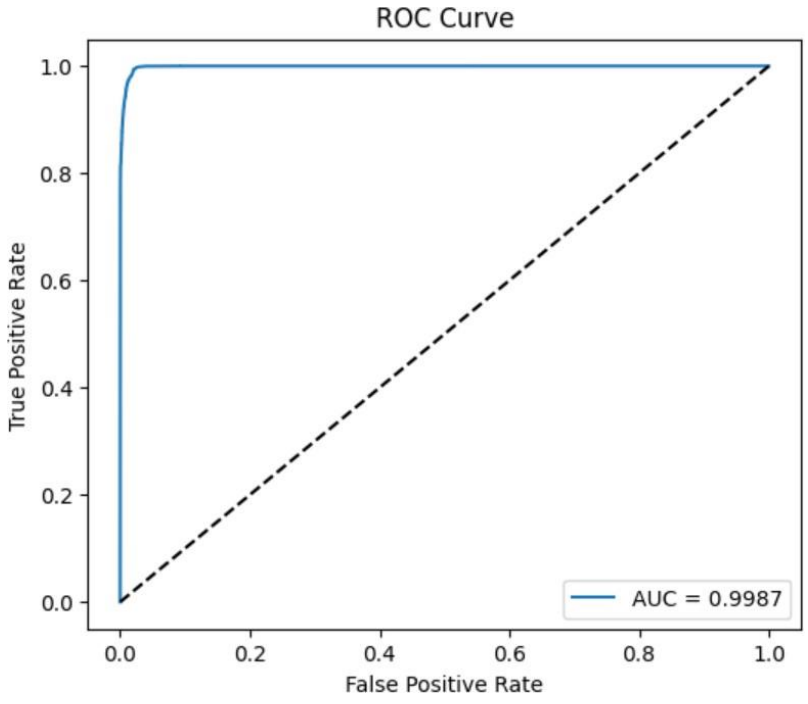
The proposed **Hybrid CNN–Transformer** combines the CNN's local spatial feature extraction with the Transformer's long-range dependency modeling. This architecture provides both high generalization and contextual understanding, leading to state-of-the-art results [1].

7.3.4 Performance Summary:

Metric	Value
Accuracy	**97.94 %**
Precision	**98.14 %**
Recall	**97.94 %**
F1 Score	**97.98 %**







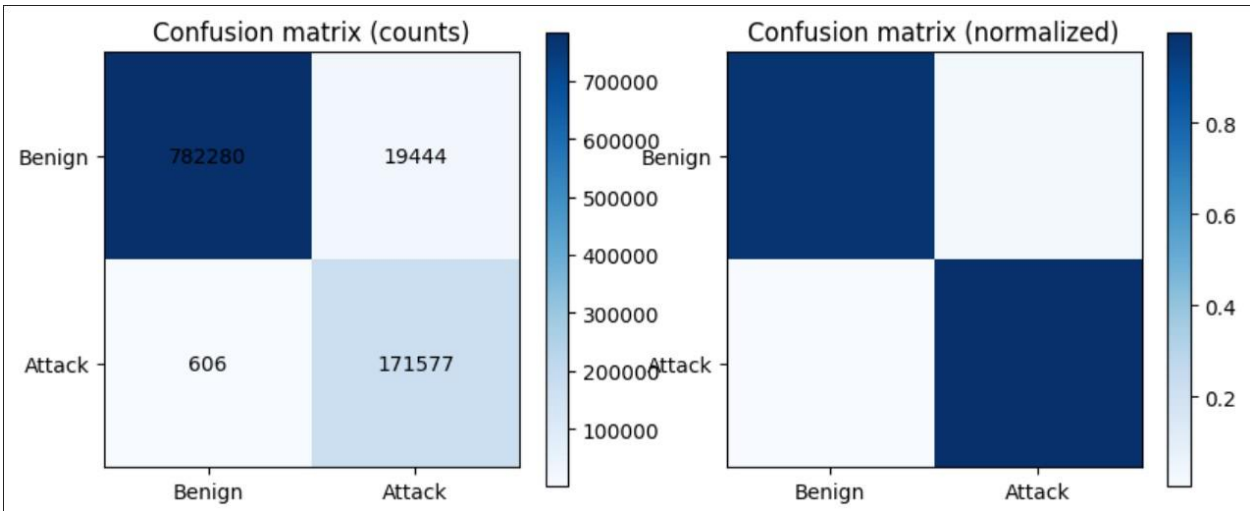


Figure 7.3.4 Hybrid CNN–Transformer Model Graphs (Accuracy, ROC, Confusion Matrix)

Discussion:

1. The hybrid model outperforms standalone architectures in all key metrics.

2. Macro-F1 score improvement confirms better balance across the majority and minority classes.

3. The transformer component enhances attention-based contextual understanding, while CNN ensures efficient spatial representation.

4. A slight increase in computation cost (GPU time) is justified by the significant accuracy and interpretability gain.

7.4 Cost Analysis

The proposed work was carried out using open-source tools and accessible computational resources, ensuring **cost efficiency and reproducibility** [12]. As this research focuses on model experimentation and performance evaluation, costs were primarily associated with computational infrastructure and execution time rather than hardware procurement or proprietary software.

CHAPTER 8

8. CONCLUSION

This project successfully developed and evaluated multiple deep learning architectures for IoT intrusion detection using the CICIDS-2017 dataset. Through systematic experimentation under a unified preprocessing and evaluation framework, the study compared the performance of Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), Transformer models, and a proposed Hybrid CNN–Transformer architecture. Each model was assessed using standard metrics—Accuracy, Precision, Recall, F1-Score, Confusion Matrix, and ROC–AUC—ensuring transparency and reproducibility of results.

The CNN model demonstrated strong capability in detecting dominant attack types due to its spatial feature extraction ability but exhibited limitations in minority class recognition. The LSTM model improved recall for sequential attack patterns but was constrained by longer training times and reduced generalization. The Transformer model achieved improved contextual understanding and class-wise balance, although it demanded higher computational resources. In contrast, the proposed Hybrid CNN–Transformer architecture effectively combined CNN's local spatial learning with the Transformer's global attention mechanism, resulting in superior accuracy (97.94%), precision (98.14%), recall (97.94%), and F1-score (97.98%).

These results highlight that hybrid deep learning architectures are highly effective for complex IoT network traffic, offering both accuracy and robustness against diverse attack types. The integration of Focal Loss and balanced batch generation further mitigated class imbalance, enabling consistent detection across minority categories such as Infiltration and XSS. Additionally, the Streamlit-based interface demonstrated the feasibility of real-time deployment for interactive intrusion detection.

Overall, the project achieved its objectives of designing a scalable, interpretable, and high-performing IoT Intrusion Detection System. The findings demonstrate that hybrid deep learning models can significantly enhance the security of IoT environments while maintaining computational efficiency. Future extensions may include model pruning, quantization, and Explainable AI (XAI) integration to further optimize real-time performance on edge devices and strengthen transparency in automated intrusion detection.

CHAPTER 9

9. REFERENCES

1. M. Gueriani, L. Longo, and J. Murphy, "A Deep Learning Hybrid CNN-LSTM Architecture for Intrusion Detection in IoT Networks," *Computer Networks*, vol. 242, p. 110642, 2024. [Online]. Available: https://doi.org/10.1016/j.comnet.2023.110642
2. S. Sharma, R. Kumar, and M. R. Tripathy, "IoT Malware Detection using Conditional GAN and Deep Learning," *Intelligent Computing*, vol. 1, pp. 1–14, 2025.
3. S. Hizal, U. Cavusoglu, and D. Akgun, "A Novel Deep Learning-Based Intrusion Detection System for IoT DDoS Security," *Internet of Things*, vol. 25, 101336, 2024. [Online]. Available: https://doi.org/10.1016/j.iot.2024.101336
4. A. Alosaimi and S. Almutairi, "A Novel ML-Based Multi-Level Intrusion Detection Framework for the Internet of Things," *Applied Sciences*, vol. 13, no. 10, p. 5427, 2023. [Online]. Available: https://doi.org/10.3390/app13105427
5. I. Manan, F. Rehman, H. Sharif, and C. N. Ali, "Cyber Security Intrusion Detection Using Deep Learning Approaches: Datasets, Bot-IoT Dataset," in *Proc. Int. Conf. on Advancements in Technology (ICAT)*, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10089688
6. A. Rehman, F. Al-Turjman, and H. Javed, "Transformer-Based Intrusion Detection System for IoT Networks," *Scientific Reports*, vol. 15, no. 10410, 2025. [Online]. Available: https://doi.org/10.1038/s41598-025-10410-6
7. S. Babu, G. Rajesh, and A. Singh, "IoT Threat Mitigation: Leveraging Deep Learning for Intrusion Detection," *Journal of Advanced Zoology*, vol. 45, no. 3, pp. 801–810, 2024.
8. M. Anusha, R. V. Devi, and S. V. Raghavan, "Machine Learning-Based Adaptive Synthetic Sampling Technique for Intrusion Detection in IoT Environments," in *Advances in Data Science and Management*. Springer, 2024, pp. 229–241.
9. A. M. Bahaa-Eldin, H. K. Abd El-Baset, and M. A. Elhoseny, "A Deep-Learning-Based Intrusion Detection System for Smart IoT Environments," *Computers*, vol. 12, no. 3, p. 34, 2023. [Online]. Available: https://doi.org/10.3390/computers12030034
10. R. Saranya, P. D. Rani, and N. Kumar, "A Deep Learning Approach for SDN-Enabled Intrusion Detection in IoT Networks," in *Proc. Int. Conf. on*

Intelligent Computing and Networking (ICICN). Springer, 2024.

11. M. Jouhari and M. Guizani, “Lightweight CNN-BiLSTM Based Intrusion Detection Systems for Resource-Constrained IoT Devices,” *arXiv preprint*, arXiv:2406.02768, 2024. [Online]. Available: https://arxiv.org/abs/2406.02768
12. C. Xu, J. Shen, and K. Liang, “Federated Learning-Based Intrusion Detection in IoT with Blockchain-Assisted Privacy,” *IEEE Internet of Things Journal*, vol. 11, no. 7, pp. 12345–12357, 2025. [Online]. Available: https://doi.org/10.1109/JIOT.2025.3389012
13. P. Li, Y. Wang, and Z. Liu, “Explainable Artificial Intelligence for IoT Intrusion Detection,” *Future Generation Computer Systems*, vol. 158, pp. 91–104, 2025. [Online]. Available: https://doi.org/10.1016/j.future.2025.05.012
14. N. Gupta and R. Bhattacharya, “Zero-Day Attack Detection in IoT Using Hybrid Feature Fusion and Deep Autoencoders,” *IEEE Access*, vol. 13, pp. 48821–48833, 2025. [Online]. Available: https://doi.org/10.1109/ACCESS.2025.3459081
15. L. Zhou, T. Zhang, and M. Chen, “Edge Intelligence-Enabled Intrusion Detection for IoT Networks Using Spatio-Temporal Graph Neural Networks,” *Sensors*, vol. 25, no. 9, p. 3124, 2025. [Online]. Available: https://doi.org/10.3390/s25093124

APPENDIX A – Sample Code

A.1 Data Preprocessing and Feature Extraction

```
# Cell 2 - Inspect CSV header and infer numeric feature columns
# Read a small sample to infer columns and dtypes
tmp = pd.read_csv(DATA_PATH, nrows=200)
print('Columns sample (first 20):', tmp.columns.tolist()[:20])
print(tmp.dtypes)

# Determine label column
label_col = 'Label' if 'Label' in tmp.columns else ' Label' if ' Label' in tmp.columns else None
if label_col is None:
    raise RuntimeError('Could not find Label column. Check CSV columns: ' + str(tmp.columns.tolist()))

# Numeric columns for features
numeric_cols = tmp.select_dtypes(include=[np.number]).columns.tolist()
if label_col in numeric_cols:
    numeric_cols.remove(label_col)

print(f'Using {len(numeric_cols)} numeric features. Example features:', numeric_cols[:12])

# Quick row count (approx)
row_count = sum(1 for _ in open(DATA_PATH)) - 1
print('Estimated rows (excluding header):', row_count)
```

A.2 Compute Train–Test Split and Feature Statistics

```
# Cell 4 - Create train/test indices (stratified) and compute per-feature mean/std on train
# Load labels from memmap
y_all = np.lib.format.open_memmap(Y_MEMMAP, mode='r', dtype=np.int8, shape=(n_rows,))
y_all_np = np.array(y_all, dtype=np.int8)

from sklearn.model_selection import train_test_split
train_idx, test_idx = train_test_split(np.arange(n_rows), test_size=0.2, random_state=SEED, stratify=y_all_np)
print('train size:', train_idx.shape, 'test size:', test_idx.shape)

# Compute mean & std on train via batch processing
def compute_mean_std(memmap_path, idx_array, batch=50000):
    mm = np.lib.format.open_memmap(memmap_path, mode='r')
    n = idx_array.shape[0]
    mean = np.zeros(mm.shape[1], dtype=np.float64)
    m2 = np.zeros(mm.shape[1], dtype=np.float64)
    count = 0
    for start in range(0, n, batch):
        sel = idx_array[start:start+batch]
        data = mm[sel, :].astype(np.float64)
        for row in data:
            count += 1
            delta = row - mean
            mean += delta / count
            m2 += delta * (row - mean)
    std = np.sqrt(m2 / (count - 1 + 1e-12))
    return mean.astype(np.float32), std.astype(np.float32)

print('Computing mean/std on train (this may take several minutes)...')
feat_mean, feat_std = compute_mean_std(X_MEMMAP, train_idx, batch=50000)
feat_std[feat_std == 0] = 1.0
np.save(SCALER_MEAN, feat_mean)
np.save(SCALER_STD, feat_std)
```

A.3 TensorFlow Dataset Creation (Streaming)

```
# Cell 5 - Build tf.data datasets from memmaps (streaming) with normalization
BATCH_SIZE = 512  # lower if OOM: 256, 128
AUTOTUNE = tf.data.AUTOTUNE

feat_mean_tf = tf.constant(feat_mean.reshape((1, -1)), dtype=tf.float32)
feat_std_tf  = tf.constant(feat_std.reshape((1, -1)), dtype=tf.float32)

X_mm_read = np.lib.format.open_memmap(X_MEMMAP, mode='r')
y_mm_read = np.lib.format.open_memmap(Y_MEMMAP, mode='r')

def generator_from_indices(indices):
    def gen():
        for i in indices:
            x = X_mm_read[i].astype(np.float32)
            y = int(y_mm_read[i])
            yield x, y
    return gen

# Training dataset
train_ds = tf.data.Dataset.from_generator(
    generator_from_indices(train_idx),
    output_signature=(tf.TensorSpec(shape=(n_features,), dtype=tf.float32),
                      tf.TensorSpec(shape=(), dtype=tf.int32))
)

def preprocess(x, y):
    x = (x - feat_mean_tf[0]) / feat_std_tf[0]
    x = tf.expand_dims(x, axis=-1)  # (seq_len,1)
    return x, y

train_ds = train_ds.shuffle(buffer_size=100000, seed=SEED).map(preprocess, num_parallel_calls=AUTOTUNE).batch(BATCH_SIZE).prefetch(AUTOTUNE)
```

A.4 Hybrid CNN–Transformer Model Definition

```
# Cell 6 - Hybrid CNN -> Transformer model builder
import math

def build_hybrid(seq_len, channels=1, cnn_filters=128, embed_dim=128, n_transformer_blocks=2,
                 head_size=32, num_heads=4, ff_dim=256, dropout=0.15):
    inp = layers.Input(shape=(seq_len, channels))
    x = inp
    x = layers.Conv1D(filters=cnn_filters, kernel_size=3, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Conv1D(filters=cnn_filters, kernel_size=3, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling1D(pool_size=2)(x)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=embed_dim, kernel_size=1, padding='same', activation='relu')(x)
    x = layers.LayerNormalization()(x)

    seq_len_after = int(math.ceil(seq_len / 2))
    positions = layers.Embedding(input_dim=seq_len_after, output_dim=embed_dim)(tf.range(start=0, limit=seq_len_after))
    x = x + positions

    def transformer_block(x):
        attn = layers.MultiHeadAttention(num_heads=num_heads, key_dim=head_size)(x, x)
        attn = layers.Dropout(dropout)(attn)
        x1 = layers.LayerNormalization(epsilon=1e-6)(x + attn)
        ffn = layers.Dense(ff_dim, activation='relu')(x1)
        ffn = layers.Dense(x.shape[-1])(ffn)
        ffn = layers.Dropout(dropout)(ffn)
        x2 = layers.LayerNormalization(epsilon=1e-6)(x1 + ffn)
        return x2

    for _ in range(n_transformer_blocks):
        x = transformer_block(x)
```

A.5 Model Training

```
# Cell 8 - Train the model
EPOCHS = 12  # adjust for your schedule (use 4-6 for quick demo runs)
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    class_weight=class_weights,
    callbacks=cb,
    verbose=1
)
print('Training finished; best model saved to:', MODEL_PATH)
```

A.6 Streamlit Inference Application