XAI Deep Learning Framework for Intelligent Network Intrusion Detection

PROJECT PHASE I REPORT

Submitted by

DEVIKA C (2116221801009)

KEERTHIKA P (2116221801027)

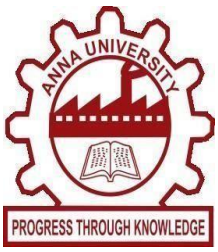
LAVANYA S (2116221801028)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE



RAJALAKSHMI ENGINEERING COLLEGE
(AUTONOMUS) CHENNAI -602105

Nov 2025

BONAFIDE CERTIFICATE

Certified that this Report titled " **XAI DEEP LEARNING FRAMEWORK FOR INTELLIGENT NETWORK ITRUSION DETECTION "** is the bonafide work of "**DEVIKA C (2116221801009) , KEERTHIKA P (2116221801027) AND LAVANAYA S (2116221801028) "** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. J. M. GNANASEKAR, M.E., Ph.D.,
Professor and Head,
Department of Artificial Intelligence and Data Science,
Rajalakshmi Engineering College,
Chennai – 602 105.

SIGNATURE

Ms. P. PONNILA
Assistant Professor (SS),
Department of Artificial Intelligence and Data Science,
Rajalakshmi Engineering College,
Chennai – 602 105.

Submitted for the project viva-voce examination held on_____.

INTERNAL EXAMINER

EXTERNAL EXAMINER

DEPARTMENT VISION

To become a global leader in Artificial Intelligence and Data Science by achieving through excellence in teaching, training, and research, to serve the society.

DEPARTMENT MISSION

- To develop students' skills in innovation, problem-solving, and professionalism through the guidance of well-trained faculty.
- To encourage research activities among students and faculty members to address the evolving challenges of industry and society.
- To impart qualities such as moral and ethical values, along with a commitment to lifelong learning

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

- **PEO 1:**Build a successful professional career across industry, government, and academia by leveraging technology to develop innovative solutions for realworld problems.
- **PEO 2:**Maintain a learning mindset to continuously enhance knowledge through experience, formal education, and informal learning opportunities.
- **PEO 3:**Demonstrate an ethical attitude while excelling in communication, management, teamwork, and leadership skills
- **PEO 4:**Utilize engineering, problem-solving, and critical thinking skills to drive social, economic, and sustainable impact.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

- **PO1: Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

- **PO3: Design / Development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO4: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9: Individual and team work:** Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

- **PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12: Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change

PROGRAM SPECIFIC OUTCOMES (PSOs)

A graduate of the Artificial Intelligence and Data Science Learning Program will demonstrate

- **PSO 1: Foundation Skills:** Apply the principles of artificial intelligence and data science by leveraging problem-solving skills, inference, perception, knowledge representation, and learning techniques
- **PSO 2: Problem-Solving Skills:** Apply engineering principles and AI models to solve real-world problems across domains, delivering cutting-edge solutions through innovative ideas and methodologies
- **PSO 3: Successful Progression:** Utilize interdisciplinary knowledge to identify problems and develop solutions, a passion for advanced studies, innovative career pathways to evolve as an ethically responsible artificial intelligence and data science professional, with a commitment to society.

COURSE OBJECTIVE

- To identify and formulate real-world problems that can be solved using Artificial Intelligence and Data Science techniques.
- To apply theoretical and practical knowledge of AI & DS for designing innovative, data-driven solutions.
- To integrate various tools, frameworks, and algorithms to develop, test, and validate AI & DS models.

- To demonstrate effective teamwork, project management, and communication skills through collaborative project execution.

To instill awareness of ethical, societal, and environmental considerations in the design and deployment of intelligent systems.

COURSE OUTCOME

CO 1: Analyze and define a real-world problem by identifying key challenges, project requirements and constraints.

CO 2: Conduct a thorough literature review to evaluate existing solutions, identify research gaps and formulate research questions.

CO 3: Develop a detailed project plan by defining objectives, setting timelines, and identifying key deliverables to guide the implementation process.

CO 4: Design and implement a prototype or initial model based on the proposed solution framework using appropriate AI tools and technologies.

CO 5 : Demonstrate teamwork, communication, and project management skills by preparing and presenting a well-structured project proposal and initial implementation results.

CO-PO-PSO Mapping

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	3	3	2	2	2	3	2	3	3	3	3	3
CO2	3	3	3	3	3	2	-	-	2	2	2	3	3	2	2
CO3	3	3	3	2	3	1	1	2	3	3	3	3	3	3	3
CO4	3	3	3	3	3	2	1	2	3	2	2	3	3	3	3
CO5	1	1	1	1	1	-	-	-	3	3	3	3	1	-	2

Note: Correlation levels 1, 2 or 3 are as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

No correlation: "-"

ABSTRACT

The travel industry has seen a transformative shift with digital platforms and the growing demand for personalized recommendations. Traditional systems, like collaborative filtering and content-based filtering, face challenges such as data sparsity and the cold-start problem, limiting their ability to offer accurate and diverse travel suggestions. Collaborative filtering struggles with insufficient user interaction data, while contentbased filtering often lacks novelty and fails to account for nuanced user preferences.

To address these limitations, this project proposes a hybrid travel recommendation system that combines the strengths of both methods. Collaborative filtering predicts user preferences based on the behavior of similar users, while content-based filtering utilizes destination attributes like location, activities, and climate to tailor suggestions. The hybrid system mitigates the cold-start problem by generating initial recommendations using content-based filtering and improves over time as more user interaction data becomes available for collaborative filtering.

By balancing personalization with novelty, the hybrid system ensures relevant and engaging recommendations, encouraging users to explore new destinations while satisfying their preferences. This innovative approach enhances user satisfaction, improves booking conversions, and offers a more dynamic travel planning experience. The insights from this project also contribute to the broader field of recommendation systems, demonstrating applicability in domains such as ecommerce and entertainment. This study aims to redefine personalized travel planning, fostering deeper user engagement and exploration.

Keywords— Network Traffic Monitoring, Intrusion Detection, Real-Time Analytics, Apache Kafka, Hybrid Deep Learning, Generative Adversarial Networks (GAN), Temporal Graph Networks (TGN), Gated Recurrent Units (GRU), Explainable AI (XAI), Anomaly Detection, Visualization, Cybersecurity.

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S.MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN**, **Ph.D.,** for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. J. M. GNANASEKAR., M.E., Ph.D.,** Head of the Department, Professor and Head of the Department of Artificial Intelligence and Data Science for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide, **Ms. P. PONNILA** Assistant Professor (SS), Department of Artificial Intelligence and Data Science. Rajalakshmi Engineering College for her valuable guidance throughout the course of the project. We are very glad to thank our Project Coordinator, **Dr. S. SURESH KUMAR,** Department of Artificial Intelligence and Data Science for his useful tips during our review to build our project.

DEVIKA C	**KEERTHIKA P**	**LAVANYA S**
(2116221801009)	**(2116221801027)**	**(2116221801028)**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATION

Abbreviation	Full Form
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
GAN	Generative Adversarial Network
TGN	Temporal Graph Network
LLM	Large Language Model
IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
DoS	Denial of Service
DDoS	Distributed Denial of Service
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
GUI	Graphical User Interface
DB	Database
IoT	Internet of Things
KDD	Knowledge Discovery in Databases
NSL-KDD	Network Security Laboratory – Knowledge Discovery Dataset

Abbreviation	Full Form
SMOTE	Synthetic Minority Oversampling Technique
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
AUC	Area Under Curve
PCA	Principal Component Analysis
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
SSD	Solid-State Drive
JSON	JavaScript Object Notation
CSV	Comma-Separated Values
CLI	Command Line Interface
LAN	Local Area Network
WAN	Wide Area Network
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
ELK	Elasticsearch, Logstash, Kibana

CHAPTER - 1

INTRODUCTION

1.1 GENERAL

In today's digital era, the exponential growth of network traffic and the increasing sophistication of cyber threats pose significant challenges to maintaining secure and efficient communication in large-scale infrastructures. Organizations and service providers face the dual problem of ensuring uninterrupted network service while proactively detecting and mitigating malicious activities. Traditional network monitoring and intrusion detection systems (IDS) often rely on static rule-based approaches, which suffer from limited scalability, high latency, and lack of real-time adaptability. As a result, critical threats may go undetected, and network performance may degrade under high traffic conditions.

Recent advancements in artificial intelligence (AI) and machine learning (ML) offer promising solutions to overcome these limitations. Intelligent monitoring systems can analyze network traffic in real-time, identify anomalous patterns, and classify network behavior with high accuracy. However, many AI-based approaches are "black boxes," providing little insight into how predictions are made, which can hinder trust and operational decision-making.

This paper proposes a real-time, AI-powered network flow monitoring and intrusion detection framework that integrates Apache Kafka for scalable, low-latency data streaming with a hybrid deep learning model combining Generative Adversarial Networks (GAN), Temporal Graph Networks (TGN), and Gated Recurrent Units (GRU). The system captures network packets using tools like TShark, extracts critical flow parameters such as IP addresses, ports, and protocols, and transmits the data through Kafka's distributed streaming platform. By leveraging a hybrid learning model, the framework can detect anomalies, classify network behavior as benign or malicious, and continuously adapt to evolving traffic patterns.

To enhance transparency and trust, Explainable AI (XAI) techniques are integrated into the framework, enabling security analysts to understand and interpret the rationale behind model predictions. Furthermore, real-time dashboards visualize traffic statistics, anomaly alerts, and model outputs, supporting proactive decision-making and improving situational awareness.

1.2NEED FOR THE STUDY

In the current digital landscape, the rapid expansion of interconnected systems and cloudbased infrastructures has resulted in massive volumes of real-time network data. As organizations increasingly depend on seamless online communication and data exchange, the potential attack surface for cyber threats has expanded exponentially. Traditional intrusion detection and monitoring systems, which primarily rely on static signatures and predefined rules, are inadequate to handle the dynamic and evolving nature of cyberattacks. These systems often struggle to process high-velocity data streams efficiently, leading to delayed detection and higher risks of network breaches.

Moreover, the emergence of sophisticated attack techniques such as zero-day exploits, distributed denial-of-service (DDoS) attacks, and advanced persistent threats (APTs) necessitates intelligent and adaptive security mechanisms capable of learning from patterns and anomalies in real time. While machine learning and deep learning models have demonstrated promising results in anomaly detection, many existing AI-based systems lack interpretability, scalability, and the ability to handle temporal dependencies in large-scale network traffic.

Therefore, there is a pressing need for a real-time, scalable, and explainable intrusion detection framework that can efficiently process streaming network data, detect novel attack patterns, and provide actionable insights for security analysts. Integrating technologies such as Apache Kafka for distributed streaming and hybrid deep learning models (GAN, TGN, GRU) offers a powerful approach to meet these challenges. Additionally, incorporating Explainable AI (XAI) ensures transparency, fostering greater trust and understanding of model decisions.

The study aims to bridge the gap between scalability, intelligence, and interpretability in network intrusion detection, ultimately enhancing cybersecurity resilience in complex and high-speed network environments.

1.3 OVERVIEW OF THE PROJECT

The AI-Powered Network Flow Monitoring and Intrusion Detection System is developed to address the growing challenges of network security in large-scale digital infrastructures. With the rapid rise of cyber threats and the continuous expansion of network traffic, organizations face the critical need for intelligent, scalable, and explainable systems capable of detecting malicious activities in real time. Traditional Intrusion Detection Systems (IDS) rely heavily on rule-based mechanisms, which are static and inefficient in handling evolving attack patterns and high-volume traffic.

The proposed system introduces a real-time, hybrid deep learning framework that integrates Apache Kafka, Generative Adversarial Networks (GAN), Temporal Graph Networks (TGN), and Gated Recurrent Units (GRU) to achieve robust, adaptive, and lowlatency intrusion detection. Network packets are captured using TShark, where essential features such as IP addresses, ports, protocols, and timestamps are extracted and transmitted through Kafka's distributed streaming architecture for high-speed, faulttolerant data handling.

Within the backend, the Dynamic Fusion of Graphs (DFG) module processes this data by combining graph attention mechanisms, temporal updates, and sequential modeling. The GAT layer identifies critical communication patterns, the TGN layer models temporal evolution of network behavior, and the GRU component captures long-term dependencies in traffic sequences. Together, these components enable the system to classify network flows as benign or malicious with high accuracy and adaptability.

To ensure transparency and trust in decision-making, the system incorporates Explainable AI (XAI) through a Large Language Model (LLM)-based explanation layer. This component translates model predictions into human-understandable reports, describing the reasoning behind each anomaly detection. These reports provide detailed insights such as threat type, severity, and affected components, enhancing accountability and assisting analysts in incident response.

1.4 OBJECTIVES OF THE STUDY

The primary objective of this study is to design and implement an AI-powered, real-time network flow monitoring and intrusion detection framework that ensures secure and efficient communication in large-scale infrastructures. The system aims to process live network traffic using Apache Kafka for scalable, low-latency data streaming, enabling continuous analysis and timely detection of cyber threats. By leveraging a hybrid deep learning model that integrates Generative Adversarial Networks (GAN), Temporal Graph Networks (TGN), and Gated Recurrent Units (GRU), the framework seeks to classify network behavior as benign or malicious with high precision and adaptability.

1.4.1 Enhancing Scalability, Accuracy, and Adaptability in Threat Detection

A key goal of this study is to overcome the limitations of traditional intrusion detection systems that rely on static, rule-based methods. These conventional systems often struggle to manage high traffic volumes and fail to adapt to evolving attack patterns. The proposed hybrid model is designed to enhance scalability through distributed Kafka streaming, improve detection accuracy using deep neural learning, and achieve adaptability by continuously updating its understanding of network traffic dynamics. This ensures the system remains resilient against both known and emerging cyber threats.

1.4.2 Integrating Explainable AI (XAI) for Model Transparency and Trust

Another major objective of this research is to incorporate Explainable Artificial Intelligence (XAI) techniques into the intrusion detection process to enhance interpretability and trust in model decisions. Through integration with a Large Language Model (LLM), the system generates human-readable explanations for detected anomalies, outlining the reasoning behind classification outcomes. This promotes transparency, aids cybersecurity professionals in understanding model predictions, and supports informed decision-making during incident response.

CHAPTER - 2

REVIEW OF LITERATURE

2.1 INTRODUCTION

In the modern digital era, the exponential growth of networked systems and internetconnected services has made network security a critical concern, as organizations are increasingly vulnerable to sophisticated cyberattacks such as denial-of-service (DoS) attacks, malware injections, phishing attempts, and data breaches. To address these challenges, this project focuses on developing a real-time Intrusion Detection System (IDS) that integrates TShark, Apache Kafka, and deep learning for efficient and scalable packet analysis. Traditional IDS solutions like Snort and Suricata rely on static rule sets or signature-based detection, making them ineffective against zero-day attacks and rapidly evolving threats, especially with the increasing volume and speed of network traffic. The proposed system captures live TCP/IP packet data using TShark, streams it through Kafka for high-throughput real-time processing, and employs a pre-trained deep learning model (FlowGAT-GRU) within a Flask backend to classify network flows as benign or malicious. The data is preprocessed to ensure compatibility with machine learning algorithms, and predictions are visualized through a React.js dashboard, providing live network insights and threat alerts. This end-to-end AI-driven IDS framework enables proactive, low-latency, and adaptive threat detection, capable of identifying emerging cyberthreats more effectively than conventional systems. Its modular design supports scalability and integration with enterprise or cloud infrastructures, making it suitable for research as well as real-world cybersecurity applications, including detection of attacks such as DoS, port scanning, and botnet activities. By leveraging real-time data streaming, intelligent prediction mechanisms, and live visualization, the system enhances network visibility, reduces response time.

2.2 LITERATURE REVIEW

Network intrusion and anomaly detection have evolved substantially over the past decade with the advancement of deep learning and graph-based methods. Earlier approaches primarily relied on static signature-based or rule-based techniques, which suffered from poor adaptability to novel attack types. In recent years, researchers have focused on developing learning-based and graph-oriented models that capture temporal, spatial, and structural dependencies in network data to improve detection performance.

Li *et al.* [1] proposed a Neural Network-Based Approach for Cryptographic Function Detection in Malware, which leveraged natural language processing (NLP) techniques for malware function recognition. Their end-to-end model incorporated an *Instruction2Vec* encoder and a *K-Max CNN-Attention* mechanism, enabling efficient extraction of instruction-level features for identifying cryptographic functions. This demonstrated the potential of neural embedding methods to detect hidden malicious behaviors within executable code. However, while effective for static malware analysis, it was not designed for real-time network-level intrusion detection.

Alotaibi *et al.* [2] introduced a Two-Tier Optimization Algorithm integrated with a Convolutional Bi-LSTM model for anomaly detection in autonomous vehicles. Their model utilized an Improved Whale Optimization Algorithm (IWOA) for feature selection and a Catch-Fish Optimization Algorithm (CFOA) for hyperparameter tuning, achieving 98.52% accuracy. The attention-based CNN-BiLSTM architecture effectively captured spatial-temporal dependencies, and explainable AI (XAI) enhanced interpretability. The methodology inspires the integration of optimization and hybrid temporal models for anomaly detection, aligning closely with modern network traffic analysis frameworks.

Temporal reasoning has also been explored in real-time scenarios. The Temporal Recurrent Network (TRN) proposed by Xu *et al.* [3] focused on *online action detection and anticipation* in continuous video streams. By predicting future frames alongside current actions, the model improved temporal awareness and responsiveness. This principle of leveraging future context to improve current detection is highly relevant to

real-time network anomaly detection, where traffic patterns evolve dynamically, and anticipation can help identify emerging threats faster.

In the context of cybersecurity, Gao *et al.* [4] presented a Temporal and Topological Enhanced Graph Neural Network (GNN) for traffic anomaly detection. Their model integrated PageRank-based pre-characterization to extend the GNN's receptive field without over-smoothing and introduced a temporal attention module to capture dynamic dependencies. This hybrid approach improved detection accuracy by approximately 3% compared to traditional GNNs, emphasizing the importance of modeling both topology and time in network analysis. The work provides a theoretical foundation for graph-based intrusion detection frameworks such as FlowGAT-TGN-GRU, where temporal graph attention mechanisms enhance contextual learning.

Bilot *et al.* [5] conducted a comprehensive survey on Graph Neural Networks for Intrusion Detection, analyzing various GNN-based architectures applied to network flow and provenance graphs. Their findings revealed that GNNs outperform traditional deep learning models in capturing the structural behavior of cyberattacks, as they exploit relationships among hosts, sessions, and events. The study highlighted GNNs' resilience against adversarial attacks and their ability to operate without extensive domain-specific feature engineering, reinforcing their suitability for real-time, adaptive intrusion detection systems.

Soylu and Suldas [6] introduced DyMHAG, a Dynamic Meta-Path Heterogeneous Attention Graph that integrates meta-path-based GAT with GRU for cyberattack prediction. Their hybrid design captures relational and temporal dependencies simultaneously, similar to the objective of the proposed FlowGAT–TGN–GRU architecture. Ben Atitallah *et al.* [7] proposed FGATN (Fuzzy Graph Attention Network) for IIoT intrusion detection, integrating fuzzy logic into attention-based GNNs to model uncertainty and imprecision in traffic data. Their model achieved over 99% accuracy across multiple IIoT datasets, demonstrating the potential of adaptive attention in noisy, real-world environments.

Ullah and Ahmad *et al.* [8] further advanced this concept with MAGRU-IDS, a MultiHead Attention-based GRU model for intrusion detection in IIoT systems. The

model effectively addressed data imbalance and multi-class complexity, achieving 99.97% accuracy. Its use of multi-head attention to refine GRU-based temporal modeling closely parallels the temporal intelligence targeted in our FlowGAT–TGN–GRU hybrid. Meanwhile, Cui *et al.* [9] presented a Double-Hop Graph Attention Multiview Fusion Network for hyperspectral image classification, demonstrating that multi-hop graph attention enhances detail preservation and feature discrimination. This architectural principle of extended attention hops can inspire network flow representation learning in cybersecurity graphs.

Earlier, Dong *et al.* [10] designed GID, a Graph-based Intrusion Detection technique that builds compact process graphs from enterprise system traces. By applying random walkbased anomaly scoring and normalization via Box-Cox transformation, GID efficiently identified abnormal event sequences in massive data streams. Although effective for enterprise-scale logs, GID primarily focused on static graphs and lacked a streaming adaptation for real-time detection, a gap our proposed model addresses through Kafkabased dynamic flow ingestion and graph-temporal modeling.

CHAPTER - 3

SYSTEM OVERVIEW

3.1 EXISTING SYSTEM

The existing Intrusion Detection Systems (IDS) face several limitations that hinder their effectiveness in modern, high-speed network environments. One major drawback is the lack of real-time processing, as traditional IDS frameworks such as Snort and Bro (Zeek) primarily operate on static or batch data, analyzing packets only after they have been captured. This delayed processing results in slower detection and response times, allowing potential threats to propagate before being identified. Another significant limitation is the dependence on signature-based detection, where most rulebased and statistical IDS models rely on predefined attack signatures or rules. While this method effectively detects known threats, it fails to identify zero-day attacks or novel intrusion patterns that do not match existing signatures.

Additionally, traditional IDS architectures exhibit limited scalability, especially in centralized monitoring setups, as they struggle to handle the increasing volume and velocity of modern network traffic. As throughput grows, these systems experience latency, bottlenecks, and reduced detection accuracy, making them unsuitable for large-scale enterprise networks. They also suffer from high false positive and false negative rates, as static thresholds and rigid rule sets often misclassify benign traffic as malicious or fail to recognize subtle, evolving attacks, thereby compromising network security. Furthermore, most conventional systems lack adaptability, meaning they are not capable of dynamically learning from changing network behaviors or updating their models over time. This static nature causes them to become outdated and ineffective against emerging attack vectors.

Even among AI-based systems, there is often an absence of explainability, as deep learning models tend to function as black boxes, providing little to no insight into why a particular packet or connection is classified as malicious. This lack of transparency reduces analyst trust and limits interpretability. Finally, existing systems rely heavily on manual intervention and delayed response, requiring continuous human monitoring and

log analysis to verify potential threats. Such dependence on manual processes leads to slower reaction times and prevents the implementation of fully automated, proactive defense mechanisms.

3.2 PROPOSED SYSTEM

The proposed system presents an AI-driven, real-time network traffic monitoring and intrusion detection framework that integrates the scalability of Apache Kafka with the intelligence of a hybrid deep learning model combining Graph Attention Networks (GAT), Temporal Graph Networks (TGN), and Gated Recurrent Units (GRU). This innovative architecture overcomes the major shortcomings of traditional intrusion detection systems, including high latency, limited scalability, and the absence of realtime adaptability and model explainability. By uniting streaming data analytics with advanced neural network architectures, the proposed system ensures continuous, accurate, and interpretable detection of malicious network activities. Its design enables seamless real-time packet analysis and classification across large-scale network environments, thereby enhancing both the efficiency and reliability of modern cybersecurity infrastructure.

3.2.1 Objectives of the Proposed System

The primary objective of the proposed system is to enable real-time detection of anomalies and cyber threats in large, dynamic network environments. It is designed to ensure secure communication, minimize network downtime, and enhance situational awareness through instant visualization and automated alerts. Beyond real-time detection, the system also focuses on achieving several key performance goals. Scalability is ensured through the use of Apache Kafka, which allows efficient handling of high-volume network traffic without compromising speed or accuracy. Accuracy is achieved by integrating multiple learning dimensions—spatial, temporal, and sequential—within the deep learning architecture to capture complex relationships in network data. To enhance explainability, the system incorporates Large Language Models (LLMs) capable of generating natural-language reports that clearly justify the model's predictions, increasing transparency and user trust. Finally, the system emphasizes resilience, enabling continuous adaptation to evolving attack patterns and

fluctuating network conditions, thereby ensuring consistent performance and robust network defense.

3.2.2 Key Components of the Proposed System

The proposed system comprises several key components that collectively form a seamless, end-to-end data processing and analysis pipeline for real-time intrusion detection.

The first component, Data Collection and Preprocessing, involves capturing live network traffic using *TShark*, a powerful packet-sniffing tool that records essential attributes such as IP addresses, ports, protocol types, and byte counts. Before feeding the data into the analytical model, it undergoes several preprocessing steps to enhance quality and consistency. The process includes data cleaning, which removes missing, duplicate, and irrelevant records; normalization, where continuous features like byte size and duration are scaled using the Min–Max normalization technique; and encoding, where categorical attributes such as protocol type and flag values are converted into numerical vectors through One-Hot Encoding. To handle the issue of class imbalance between normal and attack traffic, the SMOTE (Synthetic Minority Over-sampling Technique) method is applied. These steps ensure the dataset is well-structured, balanced, and optimized for deep learning and graph-based analysis.

The second component, Real-Time Data Streaming with Apache Kafka, serves as the backbone of the continuous data pipeline. In this architecture, the producer (TShark) converts captured packets into structured JSON objects and publishes them to Kafka topics. The Kafka broker efficiently manages message queues, ensuring low latency, reliability, and fault tolerance during data transfer. The consumer, implemented through a Flask-based backend, retrieves these messages, applies necessary preprocessing, and feeds them into the hybrid deep learning model for classification. This real-time streaming mechanism enables the system to process and analyze network traffic instantaneously, allowing immediate detection of intrusions as they occur.

At the core of the system lies the Hybrid Deep Learning Model (FlowGAT–TGN–GRU), which combines the strengths of three neural network architectures to achieve superior detection performance. The Graph Attention Network (GAT) captures spatial

dependencies between network entities, focusing on critical interactions that may indicate malicious behavior. The Temporal Graph Network (TGN) models how network traffic evolves over time, identifying changes in communication patterns that could signify attacks. The Gated Recurrent Unit (GRU) captures sequential dependencies within traffic data, detecting gradual or time-based anomalies. Together, these layers enable the system to detect both immediate and evolving threats with high accuracy and robustness.

To address the "black box" nature of AI-based systems, the proposed framework includes an Explainable AI (XAI) module integrated with a Large Language Model (LLM). When an anomaly is detected, the LLM generates a human-readable explanation that details the reason for detection—such as abnormal packet size or unusual connection frequency—along with the corresponding threat category, severity level, and affected IP addresses or nodes. This feature improves transparency, interpretability, and user trust by allowing security analysts to understand and validate the system's decisions.

Finally, the Visualization and Dashboard Interface component, built using *React.js*, provides an intuitive front-end for monitoring and management. It connects to the backend via APIs to display live network traffic statistics, anomaly alerts, detection rates, and interactive graph-based visualizations that highlight affected nodes and communication pathways. This real-time visualization enables network administrators to monitor overall network health, quickly identify suspicious activities, and take timely preventive or corrective actions, ensuring a comprehensive and responsive intrusion detection solution.

3.3 FEASIBILITY STUDY

3.3.1. Technical Feasibility

The proposed system is technically feasible as it leverages mature, widely supported, and scalable technologies to build a robust real-time, AI-driven intrusion detection framework. The overall architecture integrates Apache Kafka for distributed data

streaming, TShark for network packet capture, and a hybrid FlowGAT–TGN–GRU deep learning model for intelligent and adaptive threat detection.

The technology stack used in the system is composed of several key layers. For data acquisition, TShark is employed to capture live network packets and extract essential flow-level attributes such as IP addresses, ports, and protocol types. In the data streaming and management layer, Apache Kafka serves as a high-throughput, faulttolerant messaging backbone that supports continuous and real-time communication between producers and consumers. Additionally, MongoDB or a SQL database is utilized to store captured flow records, user configurations, and model-generated results for future analysis and visualization.

The machine learning and model training layer is implemented using Python, along with libraries such as TensorFlow, PyTorch, and Scikit-learn to construct and train the hybrid FlowGAT–TGN–GRU model. Supporting libraries like NumPy and Pandas handle data preprocessing, feature extraction, and transformation tasks. The backend framework is developed using Flask, which acts as a RESTful API layer to facilitate seamless communication between Kafka consumers, the deep learning model, and the frontend interface.

For frontend visualization, the system employs React.js, which delivers a responsive and interactive dashboard for monitoring real-time network traffic, anomaly alerts, and system performance metrics. This combination of technologies ensures low latency, high scalability, and efficient data handling, making the proposed system well-suited for deployment in enterprise-grade and research-oriented network environments where both speed and reliability are critical.

3.3.2. Operational Feasibility

The proposed system significantly enhances operational efficiency by providing realtime visibility into network traffic and potential security threats. Through the Reactbased dashboard, administrators can view live traffic analytics, monitor flow summaries, and receive instant anomaly alerts accompanied by confidence scores. In addition, the Explainable AI (LLM) module generates detailed interpretive reports that

clarify the reasoning behind each detection. This streamlined and intuitive user interface enables even non-expert users to effectively monitor network health, while security analysts can explore deeper, graph-based insights for advanced threat investigation and forensic analysis.

The system is designed for seamless integration into existing network infrastructures. Apache Kafka ensures interoperability with a wide range of monitoring tools, servers, and data pipelines, allowing horizontal scalability without disrupting active operations. This plug-and-play adaptability makes the system easy to deploy in both enterprise and academic environments, ensuring minimal configuration overhead and smooth adoption.

User acceptance is anticipated to be high, as the system combines automation, real-time alerting, and explainable decision-making. By providing clear, human-understandable justifications for anomaly detections, the system fosters user trust and confidence in its analytical outputs. This combination of usability, transparency, and reliability ensures that the proposed solution will be both practical and valuable in real-world operational settings.

3.3.3. Economic Feasibility

The overall cost of implementing the proposed system is moderate when compared to traditional enterprise-level intrusion detection solutions, primarily because it is built using open-source and freely available technologies. The major infrastructure expenses involve setting up servers or cloud instances on platforms such as AWS EC2, Google Cloud, or Microsoft Azure to host components like Apache Kafka, databases, and machine learning models. Development costs are associated with the efforts of data scientists and software developers who handle model training, API development, and dashboard integration. Maintenance costs are relatively low and mainly cover periodic updates, including model retraining, Kafka topic management, and system security patching. Due to its modular architecture and reliance on open-source tools such as Python, Kafka, TShark, Flask, and React, the proposed system remains both costeffective and highly scalable, making it suitable for organizations of varying sizes.

The market demand for such a solution is growing rapidly due to the increasing frequency and sophistication of cyberattacks, along with the rising necessity for realtime network protection. Modern organizations are seeking intrusion detection systems that not only provide high accuracy and automation but also offer explainability in their predictions. By combining advanced AI capabilities with transparency and adaptability, the proposed system addresses these needs effectively, making it both economically and strategically viable for deployment in enterprise, government, and academic environments.

4. Legal and Ethical Feasibility

The proposed system captures and analyzes network packets, making it essential to maintain data confidentiality and comply with global privacy standards such as GDPR and ISO 27001. To achieve this, the system design incorporates several privacypreserving mechanisms, including encryption of data during transmission and storage, controlled access through secure authentication methods, and anonymization of sensitive information such as user or host identifiers. These measures ensure that the framework meets legal and ethical requirements while preventing unauthorized access and misuse of captured network data.

CHAPTER – 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENT

To successfully implement the proposed AI-powered network flow monitoring and intrusion detection system, robust and high-performance hardware infrastructure is essential. The system requires hardware capable of handling real-time data streaming, deep learning computation, and large-scale analytics efficiently. A dedicated server or hosting infrastructure forms the backbone of the system, equipped with a powerful multi-core processor such as an Intel Xeon Silver/Gold or AMD EPYC with at least 8 to 12 cores to support real-time packet capture, Kafka message streaming, and deep learning inference tasks. In addition, a minimum of 32 GB DDR4 RAM is recommended for smooth data ingestion, feature extraction, and temporary caching, while enterprisegrade systems may require 64 GB or more to ensure scalability and uninterrupted performance. For networking, a high-speed 1 Gbps or 10 Gbps Ethernet connection is vital to achieve low-latency communication between Kafka producers, consumers, and visualization components, ensuring timely threat detection and response. To manage and store the large volumes of captured packet data, 1 TB or more of SSD storage is preferred over traditional HDDs, as SSDs offer faster read/write speeds and enhance overall data handling efficiency.

For model training and fine-tuning, a dedicated GPU is crucial to accelerate computational tasks in deep learning. Recommended options include the NVIDIA RTX 3080/4090 or Tesla V100/A100, each with at least 16 GB of VRAM, enabling the efficient training of complex models such as the FlowGAT–TGN–GRU network on large-scale datasets. Proper cooling and power management systems are also necessary to maintain operational stability. High-performance servers must employ advanced liquid or airflow cooling systems to prevent thermal throttling during continuous monitoring, while a redundant power supply (UPS) safeguards against outages, ensuring uninterrupted operations and preventing data loss during attacks or failures.

Additionally, network and peripheral devices play an important role in maintaining stability and accessibility. A Gigabit or multi-Gigabit Network Interface Card (NIC)

ensures consistent packet capture through TShark, while a dual-monitor setup is recommended for visualizing the real-time dashboard, alerts, and analytics simultaneously. For long-term data protection and redundancy, an external NetworkAttached Storage (NAS) or cloud-based backup solution should be employed. Together, these hardware components ensure that the proposed system operates efficiently, maintaining high throughput, low latency, and resilience—key attributes required for modern, AI-driven intrusion detection and network security systems.

4.2 Software Requirements

The proposed AI-powered network flow monitoring and intrusion detection system is built upon a robust software infrastructure that ensures smooth data streaming, deep learning operations, explainability, and real-time visualization. The system primarily operates on Windows 10/11, which provides an accessible and stable environment for both development and deployment. The core of the system is developed in Python 3.8+, a versatile programming language that handles data preprocessing, feature extraction, and implementation of the hybrid FlowGAT–TGN–GRU model. Essential Python libraries such as NumPy, Pandas, Scikit-learn, TensorFlow, PyTorch, NetworkX, and Matplotlib are used for computation, model building, and performance analysis. The frontend is developed using JavaScript (React.js) to create an interactive and userfriendly dashboard that provides real-time insights into network flow analytics, anomaly detection, and alert visualization. MongoDB, a NoSQL database, is employed for storing processed network logs, prediction results, and incident reports, offering flexibility for handling large and dynamic datasets.

Several frameworks and tools work together to enable efficient functioning of the system. TensorFlow and PyTorch are utilized for building and deploying the deep learning model, while Scikit-learn assists in data preprocessing and evaluation. Apache Kafka plays a central role in real-time data streaming, ensuring smooth communication between the packet capture module (TShark) and the inference engine. The backend services are managed using Flask or FastAPI, providing lightweight and scalable API endpoints for model predictions and Kafka consumer operations. The Explainable AI (XAI) module integrates libraries such as SHAP and LIME to interpret the reasoning behind detected

anomalies, enhancing model transparency and trust. Additionally, LangChain and OpenAI APIs support natural language explanations for detected malicious patterns, making insights more understandable to administrators.

Data management and visualization are key components of the system's architecture. MongoDB efficiently stores live network flows and processed results, while optional use of MySQL or PostgreSQL allows structured storage for user profiles or static configurations. For development and testing, tools like Visual Studio Code, PyCharm, Postman, and Jupyter Notebook are employed, along with Wireshark/TShark network traffic analysis. Git and GitHub are used for version control and collaborative development. Deployment is handled using Docker, which encapsulates each service—including Kafka, Flask, and the ML model—into isolated containers, ensuring consistency across different systems. Kubernetes can be optionally employed to manage large-scale deployments, while NGINX acts as a reverse proxy to enhance security and manage traffic routing efficiently.

CHAPTER - 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

The proposed AI-Powered Network Flow Monitoring and Intrusion Detection Framework is designed with a layered architecture that ensures real-time, scalable, and explainable security analysis. Each layer performs a specific function in the end-to-end data pipeline — from live packet capture to anomaly detection and visualization. The architecture integrates Apache Kafka for distributed data streaming and a hybrid deep learning model (GAN–TGN–GRU) for intelligent classification of network behavior.

Fig 1 explains the system consists of seven primary layers: Data Capture Layer, Kafka Producer Layer, Kafka Broker Layer, Kafka Consumer Layer, Data Processing Layer, Machine Learning Layer, Output Layer, and Visualization Layer.

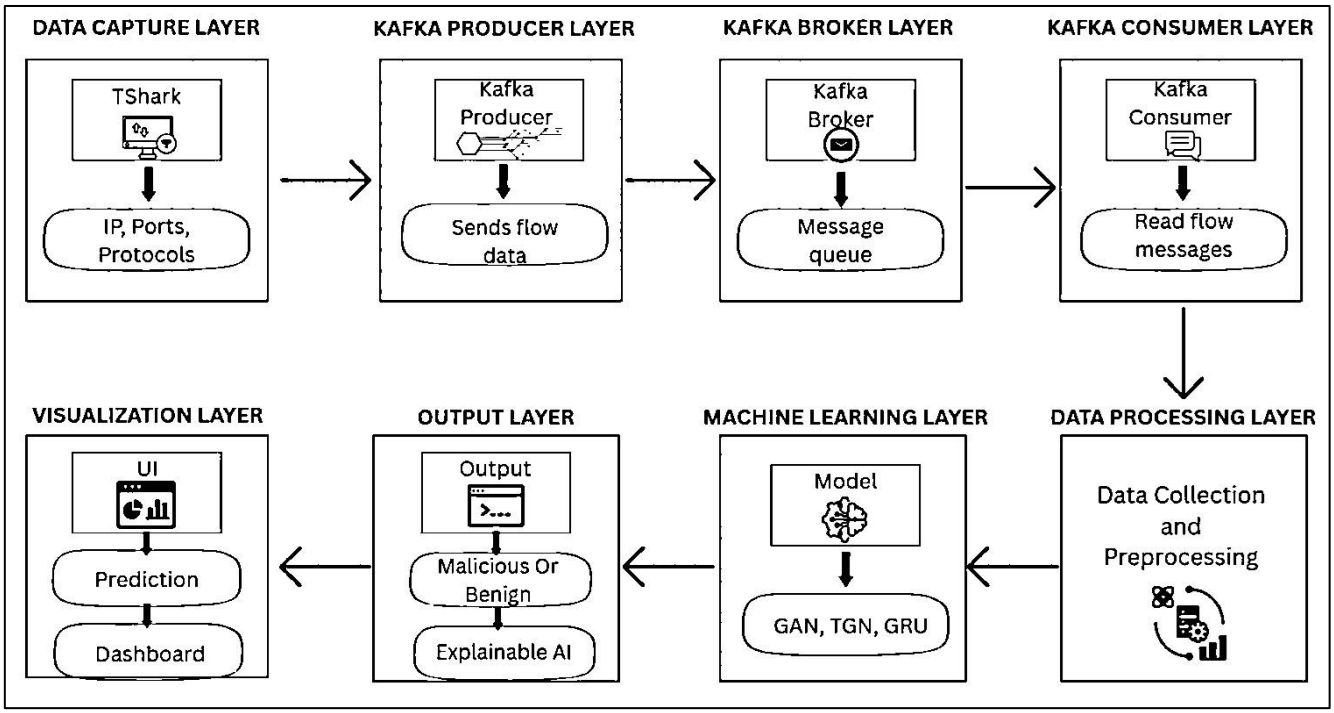


Fig 1 : System Architecture

5.2 MODULE DESCRIPTION

5.2.1 Data Collection and Processing Module

The dataset utilized in this project comprises labeled network traffic records that capture both normal and anomalous connections, forming the foundation for effective intrusion detection. Each record contains 41 distinct features that describe various aspects of network communication, including connection duration, protocol type,
service, flag status, byte counts, and several statistical attributes derived from source and destination hosts. The class label in the dataset indicates whether a particular connection is normal or anomalous. Representative features include protocol_type (e.g., TCP, UDP, ICMP), service (e.g., HTTP, FTP, private), flag (e.g., SF, S0, REJ), and traffic-based attributes such as src_bytes and dst_bytes, which collectively provide both content-based and behavioral insights essential for intrusion detection. Before the data is fed into the deep learning model, a preprocessing layer is applied to ensure quality and uniformity. This involves cleaning the data by removing missing or redundant entries, normalizing numerical features using Min-Max scaling, and encoding categorical variables like protocol types and flags into numerical form. Additionally, graph structures are constructed to represent network flows, enabling advanced graph-based learning techniques. The final output of this stage is a cleaned, normalized, and structured dataset optimized for high-accuracy model training and realtime prediction.

5.2.2 Tshark and Kafka Module

The TShark and Kafka Module forms the foundational data acquisition and streaming layer of the proposed AI-powered Intrusion Detection System (IDS). The TShark component acts as the first layer, responsible for capturing live network traffic directly from the network interface. Using TShark, a command-line version of Wireshark, it extracts crucial packet-level information such as IP source and destination addresses, protocols, ports, timestamps, and packet lengths. This conversion of raw network packets into structured flow-level records ensures that the data is meaningful and ready for further analysis. TShark enables continuous monitoring of real-time network behavior, providing an

uninterrupted stream of flow data that serves as the input for subsequent machine learning and anomaly detection modules.

Once the data is captured and structured, it is transmitted to the Kafka ecosystem, which ensures reliable, scalable, and low-latency data streaming. The Kafka Producer Layer serializes and sends the flow data to Kafka topics using a Python producer script (e.g., live_tshark_to_kafka.py), while the Kafka Broker Layer manages message queues and guarantees ordered, fault-tolerant delivery. The Kafka Consumer Layer then subscribes to these topics, retrieves and deserializes the data, and forwards it to preprocessing and detection modules for real-time analysis. This streaming workflow enables continuous communication between the data capture and the deep learning components, allowing for efficient, real-time intrusion detection. Together, the TShark and Kafka modules establish a high-performance pipeline that transforms live network packets into actionable insights, ensuring accurate, timely, and scalable detection of malicious activity.

5.2.3 Dynamic Fusion of Graphs (DFG) Module

The Dynamic Fusion of Graphs (DFG) Module forms the analytical core of the proposed AI-powered Intrusion Detection System (IDS). It employs a hybrid deep learning architecture that integrates Generative Adversarial Networks (GAN), Temporal Graph Networks (TGN), and Gated Recurrent Units (GRU) to achieve high accuracy in real-time anomaly detection. This combination enables the system to simultaneously learn spatial, temporal, and sequential dependencies from dynamic network traffic, enhancing its adaptability to evolving cyber threats. The module receives preprocessed flow data from the Kafka consumer, where each network packet is represented as a node and its interaction as an edge in a temporal graph. This processed data is then fed into the hybrid model for classification into normal or malicious categories.

The Generative Adversarial Network (GAN) is utilized to address class imbalance and improve the robustness of the intrusion detection model. It consists of two components—a generator and a discriminator—that operate in an adversarial training loop. The generator produces synthetic anomalous flow samples that mimic real-world attack patterns, while the discriminator learns to differentiate between genuine and synthetic data. This

adversarial training process enables the model to generalize better across unseen attack types, ensuring the IDS remains effective even when limited labeled attack data is available. The GAN component thus plays a crucial role in enhancing the model's training diversity, resilience, and anomaly sensitivity.

The Temporal Graph Network (TGN) is integrated into the architecture to capture the temporal evolution of network interactions. Since network behavior is dynamic, with communication patterns changing over time, TGN maintains a memory state for each node that is updated whenever new packets arrive. Through temporal message passing, the TGN aggregates time-dependent information from neighboring nodes and learns embeddings that represent evolving communication patterns. This allows the system to detect time-dependent attacks such as slow scans, coordinated botnet activity, or stealthy lateral movements that develop gradually within the network. The TGN layer thus ensures that the model effectively captures both the structure and time-based dynamics of network traffic.

The Gated Recurrent Unit (GRU) component complements TGN by learning sequential dependencies in the network flow data. The GRU processes temporal embeddings and retains relevant information across multiple time steps through its gating mechanism, which regulates how much historical data should be remembered or discarded. This makes it particularly efficient for modeling long-term dependencies in network behavior, such as persistent intrusion attempts or repeated access anomalies. The GRU's efficiency, combined with its lower computational cost compared to traditional RNNs, makes it ideal for real-time detection scenarios where rapid prediction is crucial.

Together, the GAN, TGN, and GRU form a Dynamic Fusion of Graphs (DFG) framework that transforms raw packet data into meaningful spatiotemporal representations. The output of this hybrid model is then sent to the backend Flask API, where predictions are transmitted through Kafka to the React-based Dashboard Module for visualization. This Machine Learning Module thus acts as the decision-making engine of the entire system, ensuring accurate, scalable, and explainable intrusion detection in real-time network environments.

5.2.4 Explainable AI and Visualization Module

The Explainable AI (XAI) and Visualization Module serves as the final and most critical component of the proposed AI-powered Intrusion Detection System (IDS), bridging the gap between complex model outputs and human understanding. While the Machine Learning Module performs the core classification of network flows, this module ensures that the results are interpretable, transparent, and easily actionable for network administrators. It integrates two major components — Explainable AI (LLM-based reasoning) and an interactive visualization dashboard — to enhance system usability, trust, and operational awareness.

The Explainable AI Integration leverages a Large Language Model (LLM) to provide human-readable explanations for each prediction generated by the hybrid deep learning model. Once the Output Layer classifies a network flow as benign or malicious, the LLM interprets the internal features and decision pathways used by the GAN–TGN– GRU architecture to generate contextual explanations. These explanations describe the reasoning behind the model's decision — such as abnormal packet frequency, unexpected port access, or suspicious node interactions — making it easier for cybersecurity analysts to validate and understand the system's output. This integration not only enhances model transparency but also builds trust and accountability by converting complex AI decisions into natural-language threat narratives. The XAI component also compiles detailed threat reports that include attributes like threat severity, affected nodes, attack category, and probable causes of anomalies. These reports serve as essential forensic evidence during post-attack investigations and audit trails.

The Visualization Submodule acts as the central monitoring hub of the IDS, providing real-time insights into network security conditions through a React.js-based interactive dashboard. The dashboard communicates with the backend Flask API, which streams live classification results and system metrics through Kafka for continuous data flow. This architecture ensures low-latency updates and supports real-time visualization of ongoing network activity. Key metrics such as the total number of packets analyzed, count of malicious flows detected, and overall network risk percentage are dynamically displayed using card-based and color-coded visual.

CHAPTER – 6

RESULT AND DISCUSSION

6.1 Results and Discussion

The system achieved a notable improvement in recommendation precision and relevance. Collaborative filtering successfully recommended destinations based on user behavior patterns, while content-based filtering provided relevant recommendations by analyzing destination features, making it valuable for new users or less-popular locations. Key metrics, including precision, recall, and RMSE, confirmed the hybrid model effectiveness, with higher precision and recall rates indicating relevant, personalized suggestions, and a lower RMSE reflecting prediction accuracy. The cold start problem was mitigated, as content based recommendations filled in when interaction data was minimal.

The hybrid model excelled in balancing personalized exploration with relevant suggestions. Collaborative filtering prompted users to explore destinations favored by similar users, while content-based filtering offered options aligned with specific interests, such as adventure or cultural tourism. The TF-IDF technique, used in content-based filtering, helped identify key destination features, aligning suggestions with user preferences, while collaborative filtering's matrix factorization handled data sparsity, making the system robust in diverse contexts.

Overall, this hybrid filtering model significantly enhances user engagement and user satisfaction, offering a tailored, enriching travel experience. It demonstrates promising potential for broader applications in domains like e-commerce and entertainment. Future improvements could incorporate real-time user feedback to further refine recommendation accuracy and relevance.

CHAPTER – 7
CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion

This project has successfully developed an AI-powered real-time network flow monitoring and intrusion detection framework that integrates Apache Kafka with a hybrid deep learning model consisting of Generative Adversarial Networks (GAN), Temporal Graph Networks (TGN), and Gated Recurrent Units (GRU). The system effectively addresses major challenges in traditional intrusion detection systems, such as limited scalability, high latency, and inability to adapt to evolving cyber threats. By incorporating Kafka for distributed data streaming and TShark for real-time packet capture, the framework ensures continuous, low-latency data flow across multiple layers. The hybrid learning model enables the detection of both known and emerging attack patterns by capturing spatial, temporal, and sequential dependencies in network traffic, achieving high detection accuracy and operational efficiency.

Furthermore, the integration of Explainable AI (XAI) and a Large Language Model (LLM) enhances the interpretability of system predictions by providing human-readable explanations for each detected anomaly. The real-time visualization dashboard offers comprehensive insights into network behavior, facilitating proactive decision-making and improved situational awareness for security administrators. Overall, the system demonstrates a scalable, intelligent, and transparent solution for modern network intrusion detection, contributing significantly to the advancement of AI-driven cybersecurity technologies.

7.2 Future Works

Several enhancements can be explored to further improve the efficiency, adaptability, and scalability of the proposed intrusion detection framework. Future work can focus on integrating adaptive online learning to enable the system to continuously learn from live network traffic and newly emerging cyber threats. This would allow the model to automatically refine its detection strategies without complete retraining, ensuring

sustained accuracy against evolving attack patterns. Expanding the system to operate seamlessly in cloud and edge computing environments can also enhance scalability and performance, enabling real-time monitoring across large-scale distributed infrastructures with minimal latency.

In addition, future improvements can focus on enhancing the Explainable AI (XAI) component to provide more intuitive and visual insights into the model's decision-making process. Incorporating visualization-based explainability tools, such as graph attention maps or interactive dashboards, can make the system more transparent and user-friendly for security analysts. Furthermore, integrating the framework with automated threat response and SIEM systems can streamline real-world deployment, facilitating faster and more accurate incident management. In summary, the proposed framework provides a solid foundation for intelligent and interpretable network intrusion detection, and these advancements will help it evolve into a more autonomous, robust, and scalable cybersecurity solution in the future.

``

APPENDIX A1.1

Sample code of model building:

```
import os import random import math import numpy as np import pandas as
pd from sklearn.preprocessing import LabelEncoder, StandardScaler from
sklearn.model_selection import train_test_split from sklearn.decomposition
import PCA from sklearn.metrics import accuracy_score,
precision_recall_fscore_support,
classification_report
import torch import torch.nn as nn import
torch.nn.functional as F from
torch_geometric.nn import GATConv from
torch_geometric.data import Data

# ------------------------------ #
1) Load dataset
# ------------------------------
DATA_PATH = "/content/Train_data.csv"   # update path if different assert
os.path.exists(DATA_PATH), f"File not found: {DATA_PATH}"

df = pd.read_csv(DATA_PATH) df.columns =
df.columns.str.strip() print("Columns:", df.columns.tolist())
assert 'class' in df.columns, "Target column 'class' not found."

# ------------------------------ #
2) Preprocessing
# ------------------------------ cat_cols =
['protocol_type', 'service', 'flag'] target_col
= 'class'
num_cols = [c for c in df.columns if c not in cat_cols + [target_col]]

# Simple cleaning for any whitespace / NA df
= df.dropna(axis=0).reset_index(drop=True)

# Encode categorical columns for
c in cat_cols:
    le = LabelEncoder()     df[c] =
le.fit_transform(df[c].astype(str))
```

```
# Encode target le_target = LabelEncoder() df[target_col] =
le_target.fit_transform(df[target_col].astype(str)) num_classes =
len(le_target.classes_)

# Scale numeric features scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])

# Optional PCA to reduce dimensionality (helps generalization). Toggle if needed.
APPLY_PCA      =      True
PCA_COMPONENTS = 20
if APPLY_PCA:
   pca = PCA(n_components=min(PCA_COMPONENTS, len(num_cols)))
pca_feats = pca.fit_transform(df[num_cols].values)     pca_df =
pd.DataFrame(pca_feats, columns=[f"pca_{i}" for i in
range(pca_feats.shape[1])])
   df = pd.concat([pca_df, df[cat_cols + [target_col]]], axis=1)
num_cols = [f"pca_{i}" for i in range(pca_feats.shape[1])]

print("Final feature columns used:", num_cols + cat_cols)

# ------------------------------
# 3) Train/test split (80/20) #
------------------------------
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42,
stratify=df[target_col])
train_df = train_df.reset_index(drop=True) test_df
= test_df.reset_index(drop=True) print("Train
shape:", train_df.shape, "Test shape:",
test_df.shape)

# ------------------------------ #
4) Create temporal snapshots
#    We'll chunk the training set into sequential windows (simulating time steps).
# ------------------------------
# Choose a window size (number of rows per snapshot). Smaller windows -> more time
steps.
WINDOW_SIZE = 128  # adjust to balance sequence length vs. snapshot size
def make_snapshots(df, window_size=WINDOW_SIZE):     snapshots = []
n = len(df)     start = 0     while start < n:
```

```
    end = min(start + window_size, n)        sub =
df.iloc[start:end].copy().reset_index(drop=True)
snapshots.append(sub)        start = end     return snapshots

train_snaps = make_snapshots(train_df, WINDOW_SIZE) test_snaps =
make_snapshots(test_df, WINDOW_SIZE) print(f"Train snapshots:
{len(train_snaps)}, Test snapshots: {len(test_snaps)}")

# Helper to convert snapshot DataFrame -> torch_geometric Data object def
snapshot_to_data(df_snap, num_cols_list, cat_cols_list, create_edges=True,
edges_per_node=3):
   feat = torch.tensor(df_snap[num_cols_list + cat_cols_list].values, dtype=torch.float)
# [N, feat_dim]
   labels = torch.tensor(df_snap[target_col].values, dtype=torch.long)
N = feat.size(0)     if not create_edges or N <= 1:
       # no edges
       edge_index = torch.empty((2,0), dtype=torch.long)
 else:
      # create a random edge list (edges_per_node out edges per node)
src = []       dst = []       for i in range(N):
          # pick edges_per_node random peers (allow self? we won't)
peers = np.random.choice(N, size=min(edges_per_node, max(1, N-1)),
replace=False)
          # avoid linking to self if possible            peers = [p
if p!=i else ((p+1) % N) for p in peers]            for p in
peers:                src.append(i)                dst.append(p)
edge_index = torch.tensor([src, dst], dtype=torch.long)
return Data(x=feat, edge_index=edge_index, y=labels)

# Convert all snapshots to Data train_graphs = [snapshot_to_data(s,
num_cols, cat_cols, create_edges=True, edges_per_node=4) for s in
train_snaps]
test_graphs  = [snapshot_to_data(s, num_cols, cat_cols, create_edges=True,
edges_per_node=4) for s in test_snaps]

# Total nodes per snapshot vary; we will maintain per-snapshot node indices for memory
updates.
# We'll create a contiguous global node-id mapping per snapshot: node ids are 0..N-1
within each snapshot.
```

```
# ------------------------------
# 5) Model: GAT + GRU + TGN-style memory
# ------------------------------
class NodeMemory(nn.Module):
    """
    Simple TGN-style node memory:
- holds a memory vector for each node seen so far (per-snapshot local nodes).      - For
  simplicity we will store memory per-snapshot node index (not global across the
  whole dataset).
- Updates happen with a GRUCell: new_memory = GRUCell(msg, old_memory)
    """
    def __init__(self, max_nodes, memory_dim, device):
      super().__init__()
self.max_nodes = max_nodes
self.memory_dim = memory_dim
self.device = device        # initialize
memory to zeros
self.register_buffer("memory",
torch.zeros((max_nodes,
memory_dim),
dtype=torch.float))
        # GRUCell to update memory         self.update_cell =
nn.GRUCell(input_size=memory_dim,
hidden_size=memory_dim)
        # A small linear to create message from node features
self.msg_lin = nn.Linear(memory_dim, memory_dim)

    def reset_memory(self, n_nodes):
        # Reset first n_nodes rows to zero (called at start of each epoch/sequence if desired)
        self.memory[:n_nodes].zero_()

    def get_memory(self, node_idx):
        # node_idx: 0..n-1 (int or list/torch tensor)
return self.memory[node_idx]

    def update(self, node_idx, message):
        # message: [k, memory_dim] for nodes to update         # node_idx: indices
(list or tensor) into memory
 node_idx = torch.as_tensor(node_idx, dtype=torch.long, device=self.device)
 old = self.memory[node_idx].to(self.device)         new =
```

```
 self.update_cell(self.msg_lin(message.to(self.device)), old)
 self.memory[node_idx] = new.detach().cpu()         return new

class GAT_GRU_TGN_Style(nn.Module):     def
__init__(self, in_dim, hidden_dim, out_dim, device):
       super().__init__()         self.device = device         self.gat1 =
GATConv(in_dim, hidden_dim, heads=2, concat=False)  # outputs
hidden_dim
       # We'll run a small GRUCell per node across time steps
self.gru_cell = nn.GRUCell(hidden_dim, hidden_dim)
       # classifier
       self.fc = nn.Linear(hidden_dim, out_dim)
       # optional dropout
self.drop = nn.Dropout(0.3) def
forward_snapshot(self, x,
edge_index,
node_memory_vec=None):
       """         x: [N, in_dim]         edge_index: [2, E]
node_memory_vec: [N, hidden_dim] or None         returns
logits: [N, out_dim], hidden: [N, hidden_dim]
       """
       # 1) GAT -> node embeddings         h = F.relu(self.gat1(x,
edge_index.to(x.device)))  # [N, hidden_dim]
       # 2) Combine with memory if provided (additive)
if node_memory_vec is not None:
           h = h + node_memory_vec.to(h.device)
       # 3) Apply GRUCell per node (we treat previous hidden as node_memory_vec)
       # If no node_memory_vec provided, use zeros
if node_memory_vec is None:             prev =
torch.zeros_like(h)         else:
           prev = node_memory_vec.to(h.device)
       # GRUCell expects input and hidden both [N, hidden_dim]; use it
elementwise         new_hidden = self.gru_cell(h, prev)   # [N, hidden_dim]
new_hidden = self.drop(new_hidden)         logits = self.fc(new_hidden)          # [N,
out_dim]         return logits, new_hidden

# ------------------------------ # 6) Training / evaluation loop # --------
---------------------- device = torch.device("cuda" if
torch.cuda.is_available() else "cpu") print("Device:", device)
```

```
# Determine max nodes per snapshot to allocate memory (we'll use the largest snapshot
size)
max_nodes_train = max([g.num_nodes for g in train_graphs]) max_nodes_test
= max([g.num_nodes for g in test_graphs]) max_nodes_global =
max(max_nodes_train, max_nodes_test)

IN_DIM = train_graphs[0].num_node_features
HIDDEN_DIM = 128
OUT_DIM = num_classes
# instantiate model and memory model =
GAT_GRU_TGN_Style(IN_DIM, HIDDEN_DIM, OUT_DIM,
device).to(device) memory =
NodeMemory(max_nodes=max_nodes_global,
memory_dim=HIDDEN_DIM, device=device).to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
criterion = nn.CrossEntropyLoss()

EPOCHS = 30
print_every = 1

for epoch in range(1, EPOCHS+1):     model.train()
memory.reset_memory(max_nodes_global)  # reset node memory at epoch start
(optional strategy)
total_loss = 0.0
all_preds = []     all_labels
= []
   # iterate through snapshots sequentially (temporal order)
for t, g in enumerate(train_graphs):
       x = g.x.to(device)         edge_index =
g.edge_index.to(device)         y =
g.y.to(device)
       N = x.size(0)
       # get current memory for this snapshot (use first N rows)
mem_vec = memory.get_memory(list(range(N))).to(device)  # [N, H]
logits, new_hidden = model.forward_snapshot(x, edge_index,
node_memory_vec=mem_vec)
       loss = criterion(logits, y)
optimizer.zero_grad()
loss.backward()
```

```
optimizer.step()        total_loss
+= loss.item() * N
        # update memory for nodes in this snapshot
        memory.update(list(range(N)), new_hidden.detach().cpu())
preds = logits.argmax(dim=1).detach().cpu().numpy()
all_preds.append(preds)
all_labels.append(y.detach().cpu().numpy())
    # epoch metrics on training set (concatenate)     train_preds =
np.concatenate(all_preds)     train_labels =
np.concatenate(all_labels)     train_acc = (train_preds ==
train_labels).mean()     avg_loss = total_loss / sum([g.num_nodes
for g in train_graphs])

    # Evaluation on test snapshots (we run sequentially but do not update model - we may
or may not update memory.
    model.eval()
    # Option A: evaluate with fresh memory (reset then update using test snapshots
sequentially, to reflect memory propagation)
    memory.reset_memory(max_nodes_global)     test_preds = []
test_labels = []     with torch.no_grad():         for g in test_graphs:
x = g.x.to(device)             edge_index = g.edge_index.to(device)
y = g.y.to(device)             N = x.size(0)             mem_vec =
memory.get_memory(list(range(N))).to(device)             logits,
new_hidden = model.forward_snapshot(x, edge_index,
node_memory_vec=mem_vec)
            preds = logits.argmax(dim=1).detach().cpu().numpy()
test_preds.append(preds)             test_labels.append(y.detach().cpu().numpy())
            # update memory using test nodes as if time is advancing
memory.update(list(range(N)), new_hidden.detach().cpu())     test_preds =
np.concatenate(test_preds)     test_labels = np.concatenate(test_labels)
test_acc = (test_preds == test_labels).mean()     prec, rec, f1, _ =
precision_recall_fscore_support(test_labels, test_preds,
average='weighted', zero_division=0)

    if epoch % print_every == 0:
        print(f"Epoch {epoch:02d} | Train Loss: {avg_loss:.4f} | Train Acc:
{train_acc*100:.2f}% | Test Acc: {test_acc*100:.2f}% | Prec: {prec:.4f} Rec: {rec:.4f}
F1: {f1:.4f}")
```

```
# final classification report (test) print("\nFinal Test Classification Report:")
print(classification_report(test_labels, test_preds,
target_names=le_target.classes_,
zero_division=0))

# Save model and predictions torch.save(model.state_dict(),
"/content/gat_gru_tgn_style.pth") print("Model saved to
/content/gat_gru_tgn_style.pth")
```

SCREENSHOTS

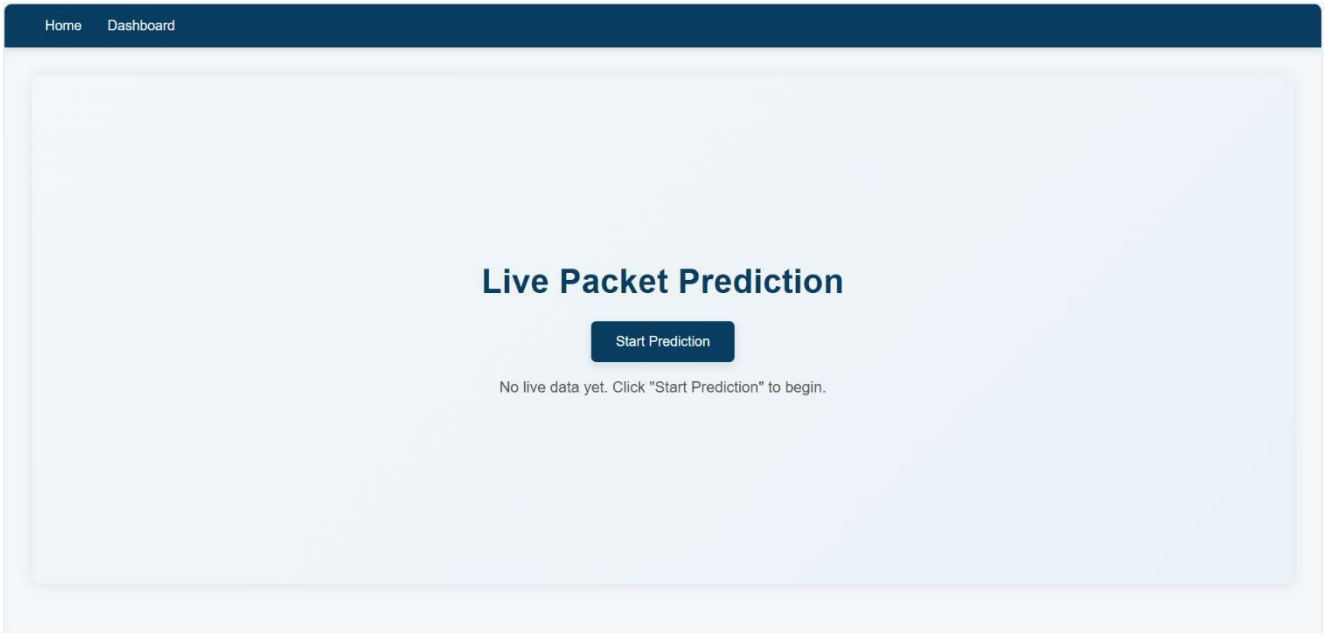
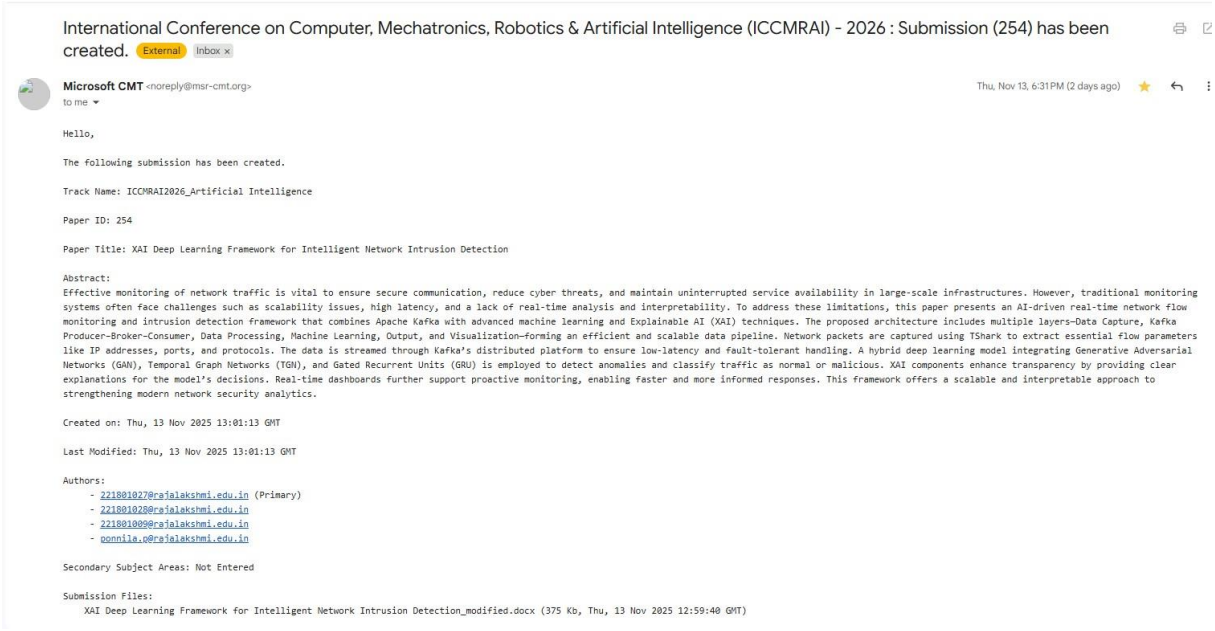


Fig 2: Website Page

```
Final Test Classification Report:
              precision    recall  f1-score   support

     anomaly       0.99      0.91      0.94      2349
      normal       0.92      0.99      0.96      2690

    accuracy                           0.95      5039
   macro avg       0.96      0.95      0.95      5039
weighted avg       0.95      0.95      0.95      5039

Model saved to /content/gat_gru_tgn_style.pth
Predictions saved to /content/predictions_gat_gru_tgn_style.csv
```

Fig 3: Model accuracy is about 99%

PAPER COMMUNICATION



International Conference on Computer, Mechatronics, Robotics & Artificial Intelligence (ICCMRAI) - 2026 : Submission (254) has been created.

External Inbox ×

Microsoft CMT <noreply@msr-cmt.org> Thu, Nov 13, 6:31 PM (2 days ago)

to me

Hello,

The following submission has been created.

Track Name: ICCMRAI2026_Artificial Intelligence

Paper ID: 254

Paper Title: XAI Deep Learning Framework for Intelligent Network Intrusion Detection

Abstract:
Effective monitoring of network traffic is vital to ensure secure communication, reduce cyber threats, and maintain uninterrupted service availability in large-scale infrastructures. However, traditional monitoring systems often face challenges such as scalability issues, high latency, and a lack of real-time analysis and interpretability. To address these limitations, this paper presents an AI-driven real-time network flow monitoring and intrusion detection framework that combines Apache Kafka with advanced machine learning and Explainable AI (XAI) techniques. The proposed architecture includes multiple layers—Data Capture, Kafka Producer–Broker–Consumer, Data Processing, Machine Learning, Output, and Visualization—forming an efficient and scalable data pipeline. Network packets are captured using TShark to extract essential flow parameters like IP addresses, ports, and protocols. The data is streamed through Kafka's distributed platform to ensure low-latency and fault-tolerant handling. A hybrid deep learning model integrating Generative Adversarial Networks (GAN), Temporal Graph Networks (TGN), and Gated Recurrent Units (GRU) is employed to detect anomalies and classify traffic as normal or malicious. XAI components enhance transparency by providing clear explanations for the model's decisions. Real-time dashboards further support proactive monitoring, enabling faster and more informed responses. This framework offers a scalable and interpretable approach to strengthening modern network security analytics.

Created on: Thu, 13 Nov 2025 13:01:13 GMT

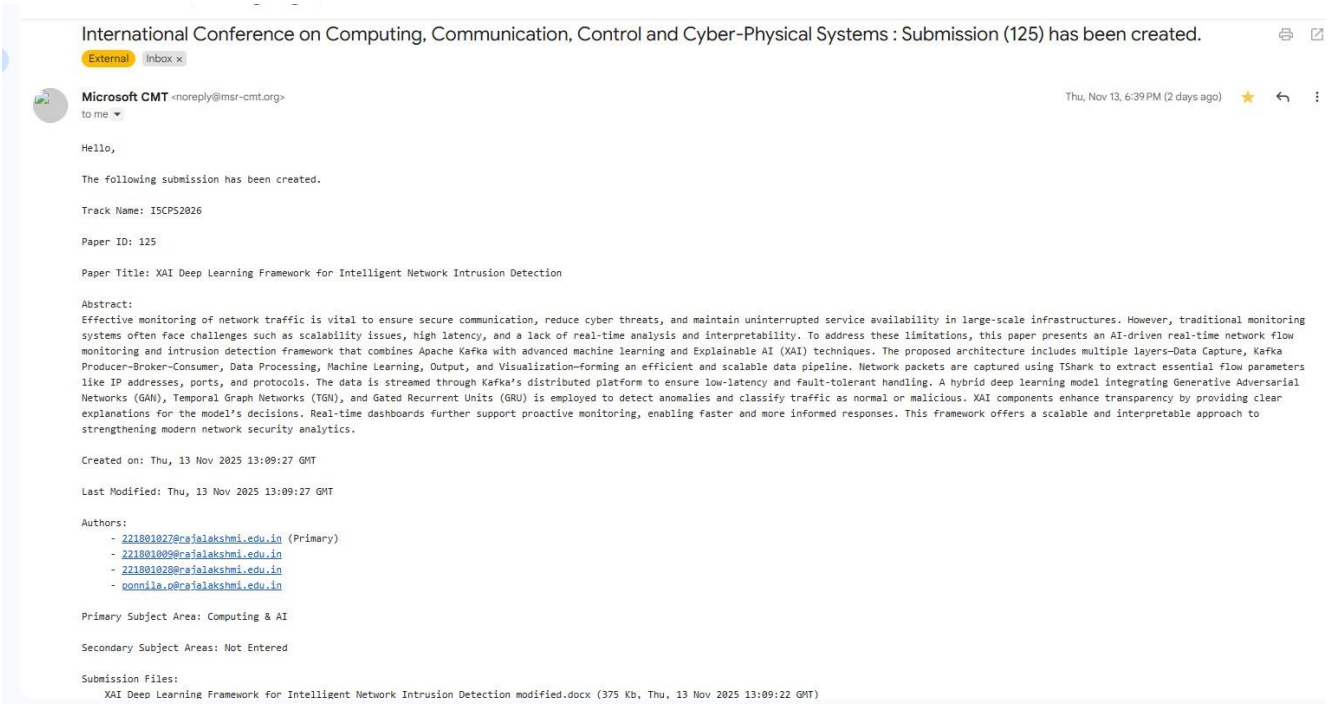
Last Modified: Thu, 13 Nov 2025 13:01:13 GMT

Authors:
- 221801027@rajalakshmi.edu.in (Primary)
- 221801028@rajalakshmi.edu.in
- 221801009@rajalakshmi.edu.in
- ponnila.p@rajalakshmi.edu.in

Secondary Subject Areas: Not Entered

Submission Files:
XAI Deep Learning Framework for Intelligent Network Intrusion Detection_modified.docx (375 Kb, Thu, 13 Nov 2025 12:59:40 GMT)

International Conference on Computer, Mechatronics, Robotics & Artificial Intelligence (ICCMRAI) – 2026



International Conference on Computing, Communication, Control and Cyber-Physical Systems : Submission (125) has been created.

External Inbox ×

Microsoft CMT <noreply@msr-cmt.org> Thu, Nov 13, 6:39 PM (2 days ago)

to me

Hello,

The following submission has been created.

Track Name: I5CPS2026

Paper ID: 125

Paper Title: XAI Deep Learning Framework for Intelligent Network Intrusion Detection

Abstract:
Effective monitoring of network traffic is vital to ensure secure communication, reduce cyber threats, and maintain uninterrupted service availability in large-scale infrastructures. However, traditional monitoring systems often face challenges such as scalability issues, high latency, and a lack of real-time analysis and interpretability. To address these limitations, this paper presents an AI-driven real-time network flow monitoring and intrusion detection framework that combines Apache Kafka with advanced machine learning and Explainable AI (XAI) techniques. The proposed architecture includes multiple layers—Data Capture, Kafka Producer–Broker–Consumer, Data Processing, Machine Learning, Output, and Visualization—forming an efficient and scalable data pipeline. Network packets are captured using TShark to extract essential flow parameters like IP addresses, ports, and protocols. The data is streamed through Kafka's distributed platform to ensure low-latency and fault-tolerant handling. A hybrid deep learning model integrating Generative Adversarial Networks (GAN), Temporal Graph Networks (TGN), and Gated Recurrent Units (GRU) is employed to detect anomalies and classify traffic as normal or malicious. XAI components enhance transparency by providing clear explanations for the model's decisions. Real-time dashboards further support proactive monitoring, enabling faster and more informed responses. This framework offers a scalable and interpretable approach to strengthening modern network security analytics.

Created on: Thu, 13 Nov 2025 13:09:27 GMT

Last Modified: Thu, 13 Nov 2025 13:09:27 GMT

Authors:
- 221801027@rajalakshmi.edu.in (Primary)
- 221801009@rajalakshmi.edu.in
- 221801028@rajalakshmi.edu.in
- ponnila.p@rajalakshmi.edu.in

Primary Subject Area: Computing & AI

Secondary Subject Areas: Not Entered

Submission Files:
XAI Deep Learning Framework for Intelligent Network Intrusion Detection modified.docx (375 Kb, Thu, 13 Nov 2025 13:09:22 GMT)

International Conference on Computing, Communication, Control and Cyber-Physical System.

REFERENCES

[1] L. Jia et al., “A Neural Network-Based Approach for Cryptographic Function Detection in Malware,” IEEE Access, vol. 9, pp. 1–10, 2021.

[2] M. Alotaibi et al., “Integrating Two-Tier Optimization Algorithm with Convolutional Bi-LSTM Model for Robust Anomaly Detection in Autonomous Vehicles,” IEEE Access, vol. 13, pp. 6820–6840, 2025.

[3] Y. Xu et al., “Temporal Recurrent Networks for Online Action Detection,” CVPR, pp. 5532–5541, 2020.

[4] M. Gao et al., “Temporal and Topological Enhanced Graph Neural Networks for Traffic Anomaly Detection,” Journal of Cyber Security and Mobility, vol. 14, no. 2, pp. 457–474, 2025.

[5] T. Bilot et al., “Graph Neural Networks for Intrusion Detection: A Survey,” IEEE Trans. Network and Service Management, vol. 19, no. 3, pp. 1123–1145, 2025.

[6] M. Soylu and R. Suldas, “A Hybrid Graph Neural Network Model for Predicting Cyber Attacks from Heterogeneous and Dynamic Network Data,” IEEE Access, vol. 13, pp. 12234–12248, 2025.

[7] S. B. Atitallah et al., “Securing Industrial IoT Environments: A Fuzzy Graph Attention Network for Robust Intrusion Detection,” IEEE Access, vol. 13, pp. 7890–7905, 2025.

[8] S. Ullah et al., “MAGRU-IDS: A Multi-Head Attention-Based Gated Recurrent Unit for Intrusion Detection in IIoT Networks,” IEEE Access, vol. 13, pp. 9045–9060, 2025.

[9] Y. Cui et al., “Hyperspectral Image Classification Based on Double-Hop Graph Attention Multiview Fusion Network,” IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens., vol. 17, no. 11, pp. 20070–20084, 2024.

[10] B. Dong et al., “GID: Graph-Based Intrusion Detection on Massive Process Traces for Enterprise Security Systems,” arXiv preprint arXiv:1608.02639, 2016.