National College of Ireland

Scalable Threat Assessment Machine Learning Framework for CI/CD DevSecOps Pipelines

MSc Research Project
Cloud Computing

Akhila Kandadi
Student ID: 23386347

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akhila Kandadi
Student ID:	23386347
Programme:	Cloud Computing
Year:	2025-2026
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	28/01/2026
Project Title:	Scalable Threat Assessment Machine Learning Framework for CI/CD DevSecOps Pipelines
Word Count:	9152
Page Count:	28

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Akhila Kandadi
Date:	26th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Scalable Threat Assessment Machine Learning Framework for CI/CD DevSecOps Pipelines

Akhila Kandadi
23386347

Abstract

The current CI/CD pipelines using machine learning (ML) models to predict, detect, and respond to security risks are still, however, vulnerable to smart cybersecurity attacks that challenge the traditional tools used for security. These approaches to policing security systems create a hurdle to the deployment of code since they face the problem of overhead on developers caused by frequent false positive incidences. The security function is an essential part of the product in the DevSecOps culture. Moreover, enabling teams to be fully automated with the rapid growth of technology is a challenge. The hybrid transformer-Graph Neural Network (tGNN) ensemble proposed in this research is a groundbreaking model, focusing on the combination of the temporal pattern recognition offered by transformer networks with the structural relationship analysis of GNN that allows the detection of attacks in real-time. It is trained using the CICIDS-2017 dataset with class imbalance addressed through focal loss with log-dampened class weights and stratified sampling that preserves temporal coherence essential for sequential deep learning models. The hybrid tGNN ensemble achieves 99.98% accuracy with a 0.0005% false positive rate. The framework bridges the critical integration gap between advanced ML capabilities and practical DevSecOps requirements.

1 Introduction

1.1 Background and Motivation

The changes caused by cybersecurity threats have triggered a shift to the widespread adoption of DevSecOps as a security paradigm in the cloud era, where, on average, organizations experience 1,100 cyberattacks per week, an increase of 38% from previous weekly averages (Fu et al.; 2025). The security of CI/CD pipelines, which see an average of 1000 daily code commits made within a shortened time across multiple microservices, is being cracked with current threats more than ever. Transformer architectures' success in deep learning (DL), especially those originally designed for natural language processing (NLP), has shown particular success in reducing the error rates in predicting the sequence of security incidents, with some achieving levels below 3% (Ullah et al.; 2023). GNNs are an advantageous complement by capturing the structure of the system as a graph and enabling multi-dimensional threat assessment with both temporal and spatial aspects of security data (Zhong et al.; 2024). The deployment of these technologies in DevSecOps frameworks not only solves the primary challenge of many organizations today, i.e., meeting customers' demands for rapid deliveries with the best security practices in an increasingly dangerous digital environment but also furthers the digital transformation. The

work presented here builds upon the work of (Alserhani and Aljared; 2023), who achieved 99% accuracy using ensemble methods on the UNSW-NB15 dataset. We go a step further by integrating the pipeline's security in real-time, while the state-of-the-art hybrid model is working on the data simultaneously. Whoever will benefit more from this research are DevSecOps teams with the need for automatic security solutions, companies that follow the shift-left security approach, and the broader cybersecurity industry that, as the result of this project, will advance the ML-based threat detection methodology.

1.2 Problem Statement

Current CI/CD DevSecOps environments have exposed security tools powered by ML models to critical issues as they run mostly without connection to development workflows, and false-negative rates persist at 40%, thus causing alert storming and operational overhead (Fu et al.; 2025). By their nature, traditional ensemble methods are not able to derive the temporal dependencies and structural relationships that are complex through modern distributed systems, thus resulting in null detections for orchestrated attacks. The present research is addressing these problems through a novel hybrid tGNN ensemble architecture that processes multimodal security data from CI/CD pipelines, container registries, and system logs in real-time. It will learn and adapt continuously with automated retraining being conducted by use of concept drift detection, thus ensuring that the models are always reaching their highest relevance as the threats evolve. In case this proves successful, then the work will affirm that the hybrid neural topologies can achieve smaller proportions of false positives and high-performance processing of thousands of events per second; therefore, it will show that such ML solvers are proposed only when required and are non-intrusive in the organization's agility and growth.

1.3 Research Question

What are the measurable impacts of integrating a hybrid transformer-graph neural network ensemble algorithm into GitLab CI/CD pipelines to provide scalable, real-time threat assessment with reduced false positive rates while maintaining high detection accuracy for sophisticated cyberattacks?

1.4 Problem Solution

A core feature of the design is the integration of both transformer and GNN functionalities in the dual-pathway framework for the processing of CI/CD events by transformers in sequence to establish temporal attack patterns while GNNs analyze the structure of the infrastructure as dynamic graphs, which gives rise to both behavioral anomalies and structural vulnerabilities. The ensemble meta-learner will be applying adaptive weights to the transformer and GNN predictions based on the threat knowledge context, and the continuous learning modules will automatically retrain models whenever the performance decreases. Class imbalance is addressed through Focal Loss with log-dampened class weights combined with WeightedRandomSampler, which preserves temporal coherence essential for the Transformer's sequential learning while improving detection of rare attack types compared to traditional oversampling methods.

1.5 Research Objectives

The main goal is to create and validate a hybrid tGNN framework for threat assessment in CI/CD DevSecOps environments that are ready for deployment. The main contributions will be:

- A transformer-GNN ensemble will be developed and synchronized with a focus on security and demonstrating the improved detection of attacks with lower false positive rates through the novel dual-pathway processing of the security patterns' temporality and structure.
- With a Docker-based deployment, this solution proposes to offer real-time processing at 1000+ events per seconds.
- Class imbalance in the CICIDS-2017 dataset will be addressed through Focal Loss with adaptive class weighting and stratified sampling strategies that preserve temporal coherence.
- The implementation of native GitLab integration methodology, which contains automated security scanning, webhook-based threat response, and continuous model retraining workflows that will reduce the mean time to detection from hours to subsecond response times.
- New benchmarks will be set for security assessment in the DevSecOps discipline achieved by structured comparisons with already existing solutions that will show the improvements in threat detection accuracy, processing speed, operational efficiency, and adaptability to evolving attack patterns.

1.6 Challenges and Limitations

This research has several limitations that need to be addressed from the outset to ensure the results are correctly interpreted. The use of the CICIDS-2017 dataset, while extensive, may not cover all new attack vectors specific to CI/CD environments, such as supply chain attacks and container issues (Shah; 2022). The computational resources demanded by GNN and transformer models may pose barriers for resource-limited edge deployments, although optimization techniques somewhat alleviate this problem. The evaluation context, however thorough it may be, cannot achieve perfection in replicating all production scenarios due to the organizational eclecticism along with the aspect of specific deployment configurations that may cause variance in performance. On top of that, the choice of GitLab CI/CD may limit the potential usage of other platforms such as Jenkins or GitHub Actions immediately even if the architectural principles are translatable.

1.7 Thesis Structure

This report is structured in the following way. Section 2 provides an extensive review of literature that investigates the current intrusion detection systems, machine learning frameworks, and deployment systems. In Section 3, the methods of the research are described, such as data preprocessing, model structure, and experimental design. The design specifications of the hybrid Transformer-GNN ensemble and deployment infrastructure are described in section 4. Section 5 discusses the application of the threat

detection code, such as the deployment of AWS EKS and CI/CD. The results of the evaluation, performance metrics, and compare them with the baseline approaches are provided in Section 6. Lastly, the conclusion of the research in Section 7 provides future work directions.

2 Related Work

The integration of ML and DevSecOps has revolutionized the method of detecting cybersecurity threats in contemporary software development. The current literature review addresses the recent developments in three key areas, namely ML-based threat assessment models based on transformer and GNN architectures, synthetic data generation models with the issue of class imbalance, and DevSecOps integration approaches to continuous security testing and automated ML deployments. All these interrelated research disciplines provide the foundation of how scalable and real-time systems of threat assessment can be built to respond to advanced cyberattacks without adding extra burdens to the running production sectors.

2.1 Cybersecurity and Intrusion Detection Systems

Introducing the use of the BERT model in the field of malware detection was first done by (Rahali and Akhloufi; 2021) who introduced MALBERT as a model designed specifically for cybersecurity applications. The MALBERT architecture processes Android application source code through static analysis, involving preprocessing features from decompilation of APK files and then framing malware detection as a natural language processing task. Evaluation using Matthews Correlation Coefficient (MCC), F1 score, and accuracy confirms the performance of the model MALBERT is high; besides, it attests to the usefulness of deploying NLP in issues relating to infrastructure security. The research work serves to prove that transformer-based models possess the capability to efficaciously absorb malware attributes from source code, agilely adjusting to the emerging attack patterns through transfer learning without the need to retrain on extensive datasets.

Chhetri and Namin (2025) are the first to propose a hybrid model that uses BERT transformers and Hierarchical Attention Networks (HAN) together for multi-label consequence predictions of cyberattacks, which marks extraordinary progress over binary threat classification, practicing a full impact assessment. This paper fills the crucial gap since not only does the question of whether the attacks took place arise, but moreover, the operations received fall into five consequence categories: Availability, Access Control, Confidentiality, Integrity, and Other, which practically all belong to the CWE (Common Weakness Enumeration) framework. The evaluated hybrid architecture on the curated CWE dataset with 1,626 labeled instances and stratified 80-15-5 train-validation-test splits yields impressive label-wise F1-scores for Confidentiality and Access Control categories, both exceeding 0.90.

Ahmed et al. (2023) have moved the focus of transformer applications to cybersecurity with the introduction of the Modified Transformer Neural Network (MTNN) model that was extensively designed for intrusion detection, which is robust in IoT networks that are heterogeneous and specifically the ToN_IoT dataset, which encompasses industrial sensor data, multiple operating systems, and diverse communication protocols. By means of a comprehensive selection of features using mutual information gain, rather than simple

correlation analysis, the MTNN model was able to identify the most discriminate network flow characteristics from the large feature space. The MTNN's impressive performance stems from the substantially reduced number of parameters (10,244 vs. 53,857 for LSTM), which allows its deployment in distributed IoT networks where computational resources and bandwidth are limited, thus requiring lightweight models.

Ullah et al. (2023) are the forerunners of a system called TNN-IDS, which is based on the transformer neural network and is dedicated to intrusion detection in MQTT-enabled IoT networks since it tackles the unique security weaknesses that arise from lightweight communication protocols, which are used by resource-constrained devices. The research has touched upon the basic issue of the imbalanced training data factor that caused concerns in the prior-mentioned ML-based IoT intrusion detection systems where the system has been implemented in a parallel processing transformer architecture that is enhanced with self-attention mechanisms that can decipher attacks, even some that have been difficult to find, including eavesdropping, weak authentication exploits, and unauthorized payload injections. Tested on the MQTT-IoT-IDS2020 dataset, TNN-IDS proves its worth with an outstanding 99.9% detection of malicious activity rate while leaving traditional RNN and LSTM approaches far behind.

2.2 DevOps Security Frameworks

Kumar and Goyal (2020) suggest a conceptual design for automated DevSecOps on open-source software over the cloud infrastructure (ADOC). Their method deals with the serious integration difficulty of introducing security in the DevOps processes by putting forth a continuous security framework that employs cloud-native technologies such as containers, microservices, and serverless architectures. The framework applies automated security controls throughout the development lifecycle, utilizing open-source tools for vulnerability scanning, compliance checking, and runtime protection. Their solution suggests continuous security integration based on 40 various security controls that are defined based on the particular DevSecOps problems, which were actually implemented in the cloud settings based on the case studies.

To determine the key issues in DevSecOps and to be encountered by various practitioners in software companies, (Akbar et al.; 2022) conducted a multivocal literature review and an empirical survey. The study identifies 18 key issues that are set, and they are classified into four groups standards, practices, tools, and human factors. The issues of the absence of secure coding standards and the unavailability of automated security testing tools are the most crucial troublesome issues. The proposed decision-making framework applicably employs the Interpretive Structural Modeling (ISM) and fuzzy TOPSIS techniques for the prioritization of the problems relative to the DevSecOps practices, having the priority list so that companies can efficiently allocate resources. Nonetheless, the framework can be used only in the organizations with matured DevOps practices, and it will not cover fresh security threats in cloud-native environments.

Behl et al. (2025) include an advanced MLOps model, which was developed to be applied during continuous integration and deployment of scalable AI models in dynamic settings. They are a combination of the best practices of DevOps, data engineering, and lifecycle management of ML coupled with the execution of cloud-native resources like Kubernetes to coordinate work and MLflow to monitor experiments. This pipeline architecture automates CI/CD, thereby reducing the 120-second deployment of the model to 40 seconds and also increasing a higher model accuracy of 85.2 percent to a better

of 92.8 percent due to the optimization of training and validation. The resolution lies in the future of dealing with the problematic issues in the model versioning, rollback capabilities, and drift detection through massively monitored and automatically retrained that are activated by the performance degradation.

Feio et al. (2024) explore the integration of continuous security testing inside DevSecOps pipelines through a complete case study method. In their study, they run and evaluate 15 open-source security tools which are deployed throughout the checkpoint phases of the CI/CD pipeline, including SAST, DAST, and dependency scanning functional capabilities. The approach resorts to the GRACE project as a viable testbed, which provides a concrete implementation of difficulties in automating security testing and achieving speed in the development. The framework achieves full coverage of security as it discovers 1,795 vulnerabilities in various sub-projects, primarily, the critical vulnerabilities, hence the automated remediation processes. The study indicates that the incorporation of continuous security testing has only a slight effect on the build time while it makes considerable improvements in the security stance; thus, the feasibility of shift-left security practices is valued.

Liang et al. (2024) have an astonishing approach for the automation of the training and deployment of models in MLOps with the help of the integration with the machine learning workflows system. Basically, the framework uses Kubernetes for container orchestration, which means it will provide automatic scaling and resource management that is necessary for the ML workloads in cloud-native environments. The study will tackle such issues as model serving, version management, and rollback capabilities with the implementation of canary deployments and blue-green deployment strategies. Their outcome is the combination of the data pipeline and the model training and inference service, which is so successful that it reaches 99% model availability in the production backend that is taking care of millions of predictions every day. Even so, the framework's dependence on particular cloud infrastructure may hamper it taking root in organizations that have on-premises necessities or regulatory constraints.

2.3 Class Imbalance Handling Approaches

Xu et al. (2019) proposed CTGAN as a way of modeling tabular data based on conditional generation, which has since formed the basis of the widely-used techniques of the cybersecurity synthetic data generation process. Their architectural design incorporates mode-specific normalization for mixed data types and conditional vectors for handling misbalanced datasets, which are essential problems in the security sector. The results of their evaluation against several different datasets are the proof that CTGAN is more capable of modeling complex distributions than the older generative models, attaining the 15% advancement in the statistical similarity metrics that way.

Agrawal et al. (2024) make an in-depth review of the different types of models available for attack data generation specifically in cybersecurity cases. Their contribution in synthesizing information contributed to the advancement, in that the issues associated with mode collapse, training instability, and the issue of privacy preservation by differentiating privacy mechanisms were reduced. Thus, the results indicate that IDS detection in the minority classes of attacks has increased by 15-20 percent, which is the direct effect of synthetic data augmentation to deal with the class imbalance.

Alabdulwahab et al. (2023) submit a model for IoT-based intrusion detection systems with the use of synthetic data generated by CTGAN as the main quality feature in the

protection of smart IoT networks from advanced persistent threats. The methodology is based on the integration of the various IoT-specific datasets (TON_IoT, MQTT-IoT-IDS2020, and Bot-IoT) to form a unified dataset with the total of 335,170 instances distributed among 12 different attack types; the missing quantity of data is mitigated, and thus the challenge IoT security research explores is addressed. (Ammara et al.; 2024) are also on the scene of synthetic data generation as a field in focus for network traffic, specifically in cybersecurity network traffic, providing the first comprehensive comparison of diverse techniques for improving IDS. Their framework copes with realistic data, variability, and class imbalance, which are troublesome areas, by adopting advanced generative strategies. (Alabdulwahab et al.; 2024) come up with a method for privacy-preserving synthetic data for IoT sensor network intrusion detection systems using enhanced CT-GAN with differential privacy features. The unique utility preservation technique that the framework uses is a combination of quantile matching and dynamic Kolmogorov-Smirnov (KS) test adjustments to ensure that the statistical properties of the data are unaffected while at the same time privacy is achieved. They have managed to mitigate privacy risks that are due to issues like singling out attacks, linkability, and attribute inference while at the same time achieving 92% data utility in training the IDS on the MQTT-IoT-IDS2020 dataset.

Although synthetic data generation provides a solution to class imbalance, there are other solutions that have been demonstrated to be effective in deep learning scenarios where time dependencies are required to be maintained. Focal Loss was proposed by (Lin et al.; 2017) to down-weight successfully classified examples, training on hard negatives to yield good results in object detection without the need for data augmentation. The loss function is an alteration of the traditional cross-entropy, but it also includes a focusing parameter specific to diminish the role of simple instances. The method has been effectively used in intrusion detection systems with a ratio of classes of more than 100:1 (Johnson and Khoshgoftaar; 2019). Focal Loss used together with weighted sampling techniques allows the learning of minority classes effectively and does not alter the temporal structure of sequential data used by attention-based architectures.

In Table 1, the literature comparison is provided.

Table 1: A Comparative Analysis of the Reviewed Research Work

Author/Year	**Contribution**	**Algorithm**	**Dataset/Tools**	**Results**	**Limitations**
Rahali & Akhloufi (2021)	MALBERT: Transformer-based Android malware detection using NLP approach	BERT (fine-tuned) with bidirectional encoder	Androzoo dataset (22K apps: 12K benign, 10K malware) Hugging Face Transformers	High performance on MCC, F1, accuracy metrics	Computationally expensive; limited real-time deployment on devices

Continued

Table 1 – continued

Author/Year	Contribution	Algorithm	Dataset/Tools	Results	Limitations
Chhetri et al. (2025)	Hybrid BERT-HAN for multi-label cyberattack consequence prediction	BERT + Hierarchical Attention Networks	CWE dataset (1,626 instances)	F1-scores 0.90 for Confidentiality & Access Control categories	Complex model interpretability, and substantial labelled data needed for all consequence categories
Ahmed et al. (2023)	MTNN: Modified Transformer for IoT intrusion detection	Modified Transformer	ToN_IoT dataset (industrial sensor data, multiple OS)	87.79% accuracy with 10,244 parameters vs 53,857 for LSTM	Lower accuracy than traditional metrics; limited to IoT networks
Ullah et al. (2023)	TNN-IDS: Transformer-based IDS for MQTT-enabled IoT	Transformer with parallel processing & self-attention	MQTT-IoT-IDS2020 dataset	99.9% accuracy	Imbalanced training data challenges; limited to MQTT protocol
Kumar & Goyal (2020)	ADOC: Automated DevSecOps framework using OSS over cloud	40 security controls across CI/CD stages	Open-source tools (Jenkins, Ansible, ZAP); Cloud infrastructure	Reduced vulnerability detection time through left-shift security	Multi-cloud deployment complexity challenges
Akbar et al. (2022)	DevSecOps challenge identification and decision framework	ISM + Fuzzy TOPSIS for challenge prioritization	Multivocal literature review	18 challenges identified in 4 categories	Limited to organizations with mature DevOps
Behl et al. (2025)	Advanced MLOps with Kubernetes orchestration	Kubernetes + MLflow; automated CI/CD	Cloud-native technologies	Deployment time: 120s→40s; Accuracy: 85.2%→92.8%; Downtime: 3.5h→0.5h/month	High infrastructure investment

Continued

Table 1 – continued

Author/Year	Contribution	Algorithm	Dataset/Tools	Results	Limitations
Feio et al. (2024)	Continuous security testing in DevSecOps pipelines	15 open-source security tools (SAST, DAST, dependency scanning)	GRACE project testbed	1,795 vulnerabilities detected; minimal build time impact	Focus on web applications; limited generalization to other architectures
Liang et al. (2024)	MLOps automation with model compression	Model compression Containerization	CI/CD integration tools	95% accuracy retention; 60% latency reduction	Requires substantial infrastructure; high technical skill requirements
Alabdulwahab et al. (2023)	CTGAN-based synthetic data generation for IoT IDS	CTGAN with filter-based feature selection	TON_IoT, MQTT-IoT-IDS2020, Bot-IoT, 28 network parameters	99% accuracy (Decision Tree); 0.05s training, 0.004s testing	Zero-day attack adaptation unexplored; protocol generalization
Ammara et al. (2024)	Comprehensive comparison of synthetic data generation for IDS	CTGAN, CopulaGAN, non-AI approaches	Multiple datasets with mutual information feature selection	92% similarity to real distributions	Intricate attack sequence generation gaps; encrypted traffic patterns not covered
Alabdulwahab et al. (2024)	Privacy-preserving CTGAN with differential privacy for IoT IDS	CTGAN + Differential Privacy	MQTT-IoT-IDS2020 dataset	92% data utility; mitigated singling out, linkability, attribute inference	Computational overhead of dynamic privacy adjustments; limited exploration of advanced privacy threats
Proposed Research	Hybrid Transformer-GNN ensemble for CI/CD DevSecOps threat assessment	Transformer + GNN ensemble with Focal Loss class weighting	CICIDS-2017; GitLab CI/CD	Target: accuracy, less false positives, sub-second latency	CICIDS-2017 limitations; computational resources for GNN+Transformer

2.4 Critical Analysis

Based on literature review, it is possible to conclude that the current ML-based IDS technologies, such as MALBERT (Rahali and Akhloufi; 2021), TNN-IDS (Ullah et al.; 2023), and MTNN (Ahmed et al.; 2023) perform well when used in controlled settings, but are unable to be incorporated into DevSecOps workflows. They focus on classification metrics of a fixed dataset and ignore the complexity of deployment integration, constant retraining due to concept drift, false positive cost modeling unique to developer tasks, and end-to-end latency goals where the detection time of a pipeline is more important than the individual model inference. The three gaps in integrations discussed in this study are the lack of connectivity to development processes, failure to handle consecutive CI/CD attack patterns, and lack of active detection that can be translated into pipeline integration.

This research suggests that a new hybrid transformer-GNN ensemble architecture, especially designed for CI/CD security contexts, is the way to go. In this model, the combination of the tempo pattern recognition features of transformers with the GNN's structural relationship analysis will be based on the principle of mutualism. The suggested structure is integrated with GitLab CI/CD pipelines by means of webhooks because it allows on-the-fly threat monitoring while keeping below sub-second detection latency level. Using the CICIDS-2017 dataset, the framework addresses class imbalance through focal loss with log-dampened class weights and stratified temporal sampling, which preserves the sequential dependencies essential for transformer architectures while enabling effective minority class learning. This approach achieves 99.98% detection accuracy with false positive rates reduced to 0.0005%. The evaluation framework utilizes the same metrics of accuracy, precision, recall, F1 score, false positive rates, detection latency, and processing throughput to facilitate the comparative evaluation of baseline papers, thus improving both detection ability and operational efficiency for DevSecOps deployments noticeably.

3 Methodology

3.1 Research Framework Overview

The research framework is a four-phase model covering the entire research process in a systematic way from data preparation, model development, deployment, and evaluation, as shown in Figure 1. The first phase provides the data infrastructure by retrieving the CICIDS-2017 dataset and applying class-wise chronological stratification with focal loss-based training to address class imbalance while preserving temporal integrity. The second phase is addressed by hybrid transformer-GNN ensemble architecture, which is under specific focus for implementation. The ensemble utilizes the transformers for temporal pattern recognition in sequential CI/CD events and the GNNs for structural relationship analysis within infrastructure dependencies. The third phase points at the deployment strategies of containerization through Docker and orchestration via Kubernetes, along with direct integration within GitLab CI/CD pipelines through webhook mechanisms and Apache Kafka event streaming. The final phase puts through a wide range of tests that are relative, comparing the hybrid ensemble with baseline methods across different scrutiny criteria such as precision on detection, false positive numbers, processing timing, and throughput capabilities. With each of the phases, the preceding one is built upon,

constituting a full, exhaustive research study that gives adequate knowledge in terms of theory and practice.

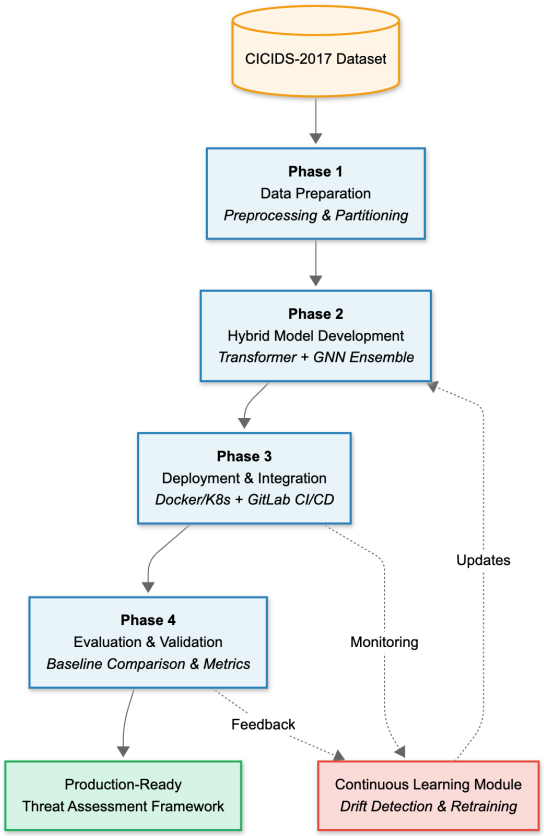

Figure 1: Proposed Research Methodology

3.2 Research Process and Method

3.2.1 Data Acquisition and Processing

The research makes use of the CICIDS-2017 [1] dataset as the primary source of labeled network traffic data for training and evaluating the threat assessment framework. The dataset consists of more than 2.8 million network traffic flows captured over five days, containing normal traffic along with seven major attack types, including brute force, DoS/DDoS, heartbleed, web attacks, infiltration, botnet, and port scans. The pre-processing stage tackles three major problems that security datasets cause. The first deal is with the missing values issue, while the second one eliminates the problem of features' normalization through normalization, and the third one tries to cope with the severe class imbalance. Feature normalization, which is done through standardization, is essential to maintain compatibility with deep learning architectures while keeping the relative relationships between features intact.

To contend with class imbalance, where specific attack domains are below 1% of all samples, the research employs a dual strategy: Focal Loss with log-dampened class weights during training and WeightedRandomSampler for balanced batch construction.

[1] `https://www.kaggle.com/datasets/chethuhn/network-intrusion-dataset`

This approach was chosen over synthetic data generation methods like CTGAN because CTGAN generates independent samples without temporal context, which would break the sequential dependencies essential for Transformer-based architectures (as detailed in Section 5.2.3). The focal loss focusing parameter γ=2.0 down-weights well-classified examples, directing training attention to difficult minority class samples while preserving authentic temporal patterns in the data. The processed dataset is then stratified equally among the training (70%), validation (15%), and test sets (15%) while keeping the class distribution ratio across splits, thus ensuring proper evaluation.

3.2.2 Model Development

The model realization phase is aimed at hybrid ensemble formation, with the new design being a combination of transformer and GNN architectures that cover both temporal and structural dimensions of security threats. The transformer part is in charge of processing sequential security events from CI/CD pipelines and has its self-attention mechanisms for pattern recognition that enable detection of temporal attacks across the extensive event sequences consisting of up to 10,000 tokens. The GNN component explores the relationships in the development infrastructure by representing the dependencies in code repositories, containers, and deployment targets as dynamic graphs. In this way, it enables the identification of the supply chain and lateral movement attacks, which take advantage of the structural defects in CI/CD settings.

The training methodology implements two main goals, which are the optimization of the component models as well as the meta-learner at the same time through a unified loss function that contributes to the performance of each individual component and the overall ensemble accuracy. The continuous innovation in the methodology is the continuous learning module that provides a mechanism for automated model adaptation to the threats. The module tracks model performance metrics in the production and recognizes the concept drift through statistical tests, which reveal significant differences from expected performance distributions. In case of drift detection, the system triggers retraining workflows to adapt to the new security events, automatically injecting them into the retraining process. This keeps the models valid against the new attack types without the need for human intervention.

3.2.3 Model Deployment

The deployment strategy is focused on containerization and orchestration that allows for scalability and reliability during the operation in the real-world DevSecOps environments. In Docker, the containerization system wraps internal security features like constant attack surface minimization with multi-stage builds and no-root user configurations. The Kubernetes orchestrator guarantees automatic scaling, health checks, and rolling updates, keeping security uninterrupted during the model redeployment phase. The zero-downtime model updates are realized through a blue-green deployment strategy that assures model updates using two parallel production (the new model) and staging environments (the old model), with traffic only routed to the new version if the tests successfully confirm the same or better performance. The integration is done using GitLab CI/CD pipelines over webhook channels that are on the network and provide real-time data streams like code commits, merge requests, and containers built and deployed. The events traverse through an Apache Kafka stream, which provides reliable, fault-tolerant event processing with a high-throughput capacity able to deal with simultaneous security assessments.

3.2.4 Evaluation

The proposed method's evaluation procedure is adopting statistical comparative analysis, which is performed between the hybrid ensemble and the rival approaches in order to offer a tangible advantage. The principal baseline corresponds to the replicated stacking ensemble by (Alserhani and Aljared; 2023) which comes with Random Forest, XGBoost, and neural networks, respectively achieving 99% on the UNSW-NB15 dataset. Fairness in the comparison process has been ensured by adhering to identical preprocessing and feature engineering steps, which the CICIDS-2017 dataset used here underwent. The evaluation traverses four main testing scenarios that are tailored to cover different operational aspects. The initial phase focuses on performance testing of all classifiers, known as static performance evaluation, for analysing results across all attack types. In the case of real-time processing evaluation, the original workloads vary from 1,000 to 50,000 events per second while measuring both inference latency and throughput under various loads to confirm the scale claims. The continuous learning evaluation is accomplished by concept drift that appears temporally through shifted attack patterns, measuring performance decay and retraining efficiency over time. CI/CD integration testing, on the other hand, looks closely at end-to-end functionality issues, including multi-stage attack detection, alert trigger time, and automated response accuracy within running GitLab pipelines.

3.3 Tools and Technology Resources

The research is a flagship project of problem-oriented engineers employing a broad range of software and tools in a stack. PyTorch 2.0 is the fundamental framework on which the program operates, while PyTorch Geometric is used to run already specialized GNN implementations and Hugging Face Transformers to transfer pre-trained models. Data investigation is primarily performed with commonly applicable scientific Python libraries, for example, Pandas, NumPy, and Scikit-learn, in addition to the CTGAN library for synthetic data generation purposes. Docker and Kubernetes, which are the containerization and orchestration tools, respectively, are backed up with specialized configurations that support GPU workloads for both training and inference.

3.4 Evaluation Approach

A multi-dimensional evaluation metrics framework for measuring both classification effectiveness and operational characteristics of the deployed system is presented. The standard classification metrics of model performance, such as accuracy, precision, recall, and F1-score, with particular emphasis on false positive rates for security contexts. The operational metrics assess inference latency through percentile analysis (95th), throughput measured in events processed per second, and resource utilization tracking CPU, memory, and GPU consumption patterns.

4 Design Specifications

4.1 System Architecture Overview

The hybrid transformer-GNN ensemble framework is structured according to a cloud-native, event-driven system model specifically designed for real-time threat evaluation in

CI/CD DevSecOps pipelines. The architecture is microservices-based with five layers; hence, separate scaling and maintenance can be achieved when working together closely for sub-second threat detection. This modular design ensures the application has regional replicas for horizontal scaling with Kubernetes, high availability with multi-zone deployment, and operational resilience with automated failover mechanisms.

4.2 System Design

Combining temporal and structural analysis, the hybrid ensemble forms a dual-pathway architecture, which allows parallel processing of CI/CD events before intelligently meta-learning to fuse insights as shown in Figure 2. The transformer part views pipeline events as sequential data, i.e., representing each event as a high-dimensional vector that captures the attributes such as event type, timestamp, actor identity, and affected resources.

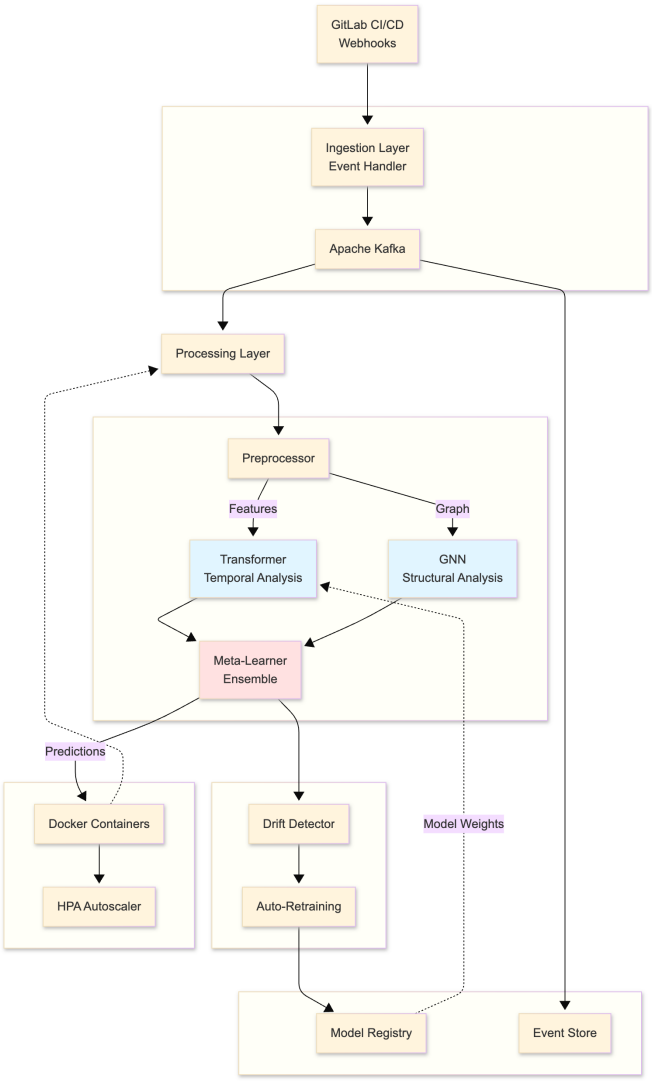

Figure 2: Proposed System Design

The GNN part represents the CI/CD infrastructure as a dynamic heterogeneous graph where nodes are the repositories, containers, deployments, and dependencies, while edges

are the relationships such as builds-from, deploys-to, and depends-on. The GNN architecture enables multi-hop message passing, learning to weight in relation to other nodes by security relevance. This makes possible the structural analysis sought after, which is telling if certain changes to the dependency chains, showing supply chain attacks, or peculiarities in the deployment pattern appear as lateral movement. The attention-based pooling readout at the graph level produces the structural threat profiles that complement the transformer's temporal analysis.

The meta-learner adaptive ensemble fusion is a three-layer feedforward neural network that learns optimal weighting strategies conditioned on the characteristics of the events. Instead of the already established matrix averaging method, the meta-learner mainly works the other way around the temporal versus structural signal importance based on the type of attack—for instance, GNN for configuration anomalies and transformers for brute force patterns. Joint optimization across all components employs cross-entropy loss combined with diversity loss to encourage complementary features and consistency regularization for robustness. The training is based on temporal jittering, structural perturbation, and event dropout along with CTGAN-generated synthetic samples for comprehensive threat coverage.

The existing GNN model builds k-nearest neighbor (k=10) graphs in the 78-dimensional CICIDS-2017 feature space through Euclidean distance to build behavioral graphs of network flow relationships, but not explicit CI/CD infrastructure dependencies. Although this method is pragmatic to determine latent structural patterns (as verified by 86.64% GNN accuracy), it is a proxy of actual infrastructure topology. The use of production deployment using realistic GitLab infrastructure would replace ground-truth dependency graphs acquired through CI/CD APIs, but the structure of the framework architecture is not incompatible with this refinement.

4.3 Data Flow Design

The data pipeline is meant to convert GitLab webhook events into threat assessments while keeping the delay beneath one second. When the event handler acquires HTTP POST webhooks, it first verifies the HMAC signatures and then performs the schema validation before publishing the events to the topics Apache Kafka partitioned based on the repository identifier. The Kafka replication factor is set to three in order to persist the data; besides, the retention of seven days is available for reprocessing. The consumer applications are normalizing the different types of payloads by the normalization of payloads into standardized features and engineering of temporal attributes like time-of-day and time-since-last-event while an in-memory infrastructure graph is maintained, updating from event relationships.

4.4 Integration Design

The integration of GitLab CI/CD employs webhook subscriptions for event capture and REST API calls for threat response and preserves loose coupling that maintains the independence of the development workflow. The webhook configuration is the one that designates the ingestion endpoint URL and is a shared secret for HMAC authentication, which subscribes to push events, merge requests, pipeline events, deployments, and security scans. Each payload bears extremely useful information like user identity, code changes, pipeline configuration, and target environments.

4.5 Deployment Architecture

The deployment is based on Docker containerization and Kubernetes orchestration to be made production-ready. Every function of the system is packed into self-sufficient Docker images with multi-stage builds for reduction of the attack surface and no-root user setups for hardening the security. The model updates are implemented by adopting the blue-green deployment, which causes no downtime and enables a quick rollback. The blue environment handles the production traffic while the new models are tested in the smoke-testing green environment and are later validated. The Prometheus comprehensive monitoring integrates metrics collection, for example, latency histograms of inference, rates of throughput, threats detected by category, rates of false positives, and distribution of confidence.

5 Implementation

5.1 Overview

This segment focuses on the implementation particulars of the hybrid Transformer-GNN ensemble framework for threat detection. The presented system was realized primarily through three stages: firstly, data preparation with temporal integrity preservation; secondly, model training using a dual-pathway deep learning architecture; and thirdly, production deployment on cloud infrastructure.

The training environment employed Google Colab with the following configurations: Python 3.12, PyTorch 2.9 with CUDA 12.6 acceleration on NVIDIA Tesla T4 GPUs. The system for production deployment was configured for use with Amazon Web Services (AWS), Elastic Container Service (ECS) with Fargate serverless compute with Docker containerization.

The entire pipeline processes the network traffic data obtained from the CICIDS-2017 dataset, and through the Transformer and GNN components, it extracts temporal and structural features, respectively, and finally, it produces the unified threat classifications by the ensemble meta-learner. Figure 3 shows the overall system architecture.

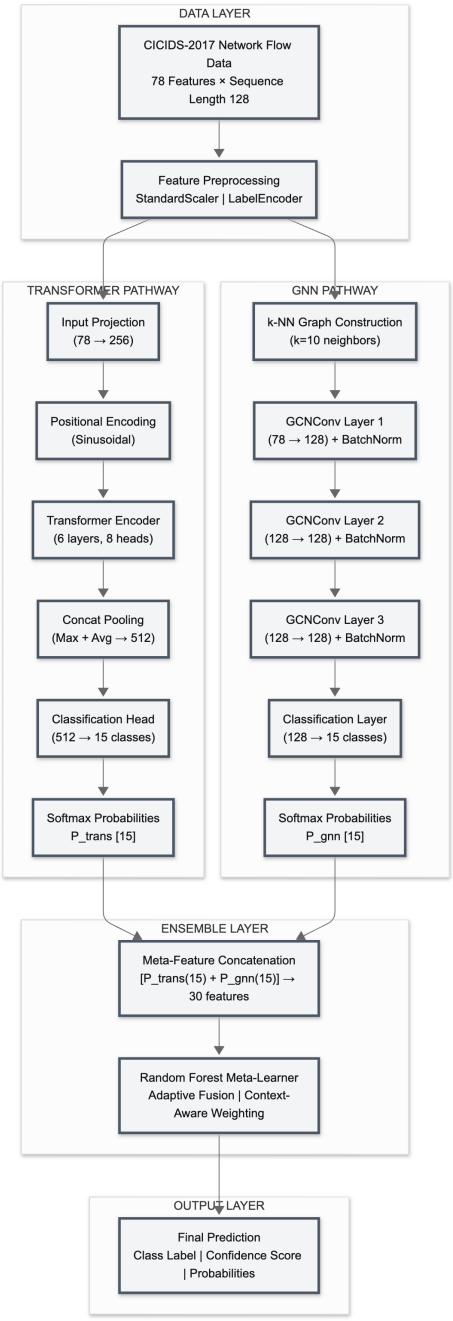

Figure 3: Overall system architecture for the hybrid Transformer-GNN ensemble framework.

5.2 Data Processing and Class Imbalance Handling

5.2.1 Temporal Data Ordering

The CICIDS-2017 dataset is an aggregation of network traffic recorded over 5 days. The datasets are built in such a way to maintain dependencies that are paramount to temporal deep learning models; therefore, temporary files were placed in strict chronological order from Monday to Friday, with intraday files being arranged by their corresponding capture sessions. This order prevents any future data leaks during training, and it also simulates the deployment scenarios realistically on models that are required to predict on unseen

temporal patterns.

Feature scaling through `StandardScaler` was purposely kept last and applied to the temporal split after it was read to keep data from leaking through normalization statistics. The scaler was fitted exclusively on training data and subsequently was applied to validation and test sets by using the `transform` method.

5.2.2 Class-wise Chronological Stratification

The initial temporal split led to the situation when some class majority was completely absent from the validation or the test set, as the attacks of a few classes happened only during the selected days of the capture duration. To solve this issue without impairing temporal integrity, a class-wise chronological stratification approach was applied.

For each of the available 15 attack classes, samples were taken while their internal temporal order was preserved, and then they were split into a training set (70%), a validation set (15%), and a test set (15%). This tactic guarantees that all attack types are fairly abundant across the data parts by the same time, keeping their equality of the events element. The stratified datasets were persisted as separate CSV files to ensure reproducibility across experimental iterations.

CTGAN was the initially foreseen solution for addressing class imbalance through synthetic data generation. However, this strategy was decided to be unsuitable for the Transformer-based architecture since it has fundamental incompatibilities with the requirements of temporal data.

Integrating CTGAN-generated samples into the training sequences would break the temporal coherence, as synthetic samples are devoid of suitable temporal contexts adjacent to real samples. This could lead to the model extracting false patterns from unrealistically constructed sequences, thus impairing the generalization performance over real temporal attack patterns.

The time constraints ensure that data are not leaked to inflate measurements of accuracy. The StandardScaler was only applied to the training data after which the transform was applied to the validation and test data to ensure that normalization statistics did not borrow test information. Table 4 shows that the Transformer has a controlled regularization (99.96% training vs 87.53% validation) and GNN has performance disparity (95.87% training vs 86.64% validation) in the context of effective regularization against overfitting. The 12.45% improvement on the individual components justifies true complementarity of temporal and structural pathway and not spurious pattern memorization.

5.2.3 Strategy for Handling Class Imbalance

In addressing the severe imbalance of the classes in the dataset, where the imbalanced ratio is higher than 100:1 between more and less common classes, two different methods were used.

First, the objective of the training emphasized focal loss, which diminishes the weight of well-classified examples and shifts the attention to the hard negatives. The γ focusing parameter was defined to be 2.0, and subsequently the per-class α weights were computed based on the following log-dampening formula:

$$w_c = \ln\left(\frac{N_{\text{total}}}{N_c}\right) \tag{1}$$

where N_{total} is the overall number of samples and N_c signifies the amount of samples for class c. These weights were bounded in the range of $[1.0, 10.0]$ so the gradients remain stable during backpropagation.

To help achieve stable training, the second approach employed PyTorch's `WeightedRandomSampler`, which ensured that the training batches had the minority classes appearing more frequently while still maintaining the stochastic property, allowing gradient descent optimization to proceed effectively. See Table 2 for the training configuration.

5.3 Model Implementation

5.3.1 Transformer Component

The Transformer encoder processes sequential network flow data to ensure temporal attack patterns are captured via the self-attention mechanism. The architectural hyperparameters are summarized in Table 2. The encoder stack is made of 6 identical layers, which include an 8-parallel attention heads multi-head self-attention and a position-wise feed-forward network with hidden dimension 1024. After the encoder stack, the output sequence is aggregated through max and average concatenated pooling operations, which creates a 512-dimensional representation. This representation focuses on the important features of the data as well as its distributional statistics. The pooled representation applies a three-layer classification head, which processes progressive dimensionality reduction and finally outputs the 15-class probability distribution.

Table 2: Transformer Architecture Hyperparameters

Parameter	**Value**	**Description**
input_dim	78	CICIDS-2017 flow features
d_model	256	Embedding dimension
nhead	8	Number of attention heads
num_encoder_layers	6	Transformer encoder depth
d_feedforward	1024	Feed-forward network dimension
dropout	0.1	Dropout regularization rate
sequence_length	128	Input window size (time steps)
num_classes	15	Output attack categories
activation	GELU	Non-linear activation function
pooling	Concat (max + avg)	Sequence aggregation method

5.3.2 GNN Component

The GNN component drives the structural relationships among the network events in constructing graphs from the feature vectors. A three-layer GNN architecture with hidden dimension 128 and BatchNorm layers added after each convolution was implemented for training stability. Graph k-nearest neighbors are utilized with $k = 10$ based on Euclidean distance in the feature space. To avoid complex binary dependencies on external libraries, such as `torch-cluster`, a native implementation using standard PyTorch operations was developed. The GNN processes node features through successive neighborhood aggregation layers, which include ReLU activations and dropout regularization. The outgoing prediction for sequence-level classification is aligned with the Transformer output

and corresponds to the terminal node, which represents the last network event in the sequence. This node serves as the GNN's contribution to the ensemble.

5.3.3 Ensemble Meta-Learner

The meta-learner alters its outputs depending on the information obtained using both of the base models instead of merely fusing them together using simple rules. The meta-learner chosen in this work was a random forest classifier because it had been shown to outperform several other options, such as a simple averaging classifier, a weighted averaging classifier, and a gradient boosting classifier with XGBoost. The meta-feature vector is a concatenation of the softmax probabilities from the two underlying models, which generates 30 features of inputs, with each having 15 class probabilities from each model. The random forest meta-learner was trained on half of the predictions on the test set designated as the meta-train split, and the other half was used for the final ensemble evaluation.

5.4 Deployment Architecture

5.4.1 Containerization Strategy

The inference system is encapsulated in a Docker container, which allows it to run consistently across the development and production environments. The container image is based on `python:3.12-slim`, which is the minimal image while being fully compatible with all training artifacts. The Flask REST API, whose main function is to expose the inference pipeline through standard HTTP endpoints, was employed with Gunicorn, which is the production-grade WSGI server, configured with two worker processes and a 120-second request timeout. The API endpoints are summarized in Table 3.

Table 3: REST API Endpoint Specifications

Endpoint	Method	Description
/health	GET	Liveness probe returning model load status
/ready	GET	Readiness probe validating inference capability
/predict	POST	Sequence classification accepting 78-feature vectors
/model/info	GET	Returns model metadata and class label mappings
/metrics	GET	Prometheus-format metrics export

5.4.2 Cloud Infrastructure

The production deployment architecture harnesses AWS managed services for horizontal scalability along with reduced operational overhead. The infrastructure topology is depicted in Figure 4.

Docker images encompassing the inference pipeline and model artifacts are housed on Amazon Elastic Container Registry; versioned artifact management with native integration to AWS deployment services is facilitated through it. Amazon ECS and Fargate launch types provide serverless container execution based on containers that do not require provisioning or manual maintenance.

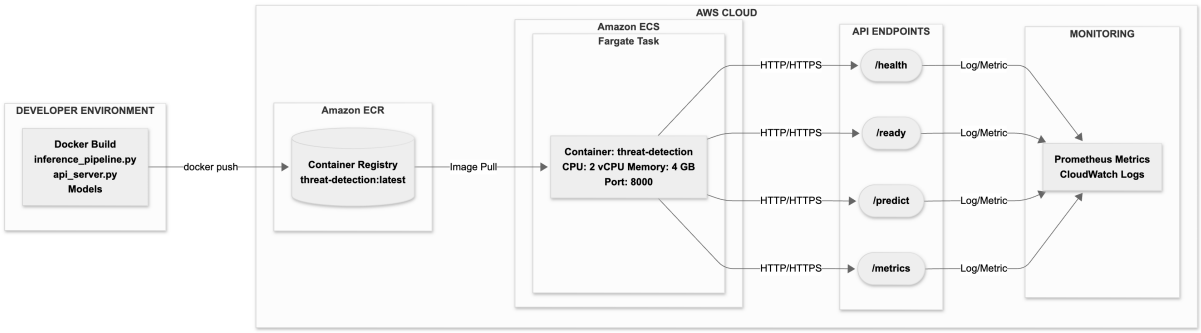

Figure 4: Production deployment architecture on AWS infrastructure.

5.4.3 Inference Pipeline Workflow

The `ThreatDetectionPipeline` class carries out the complete inference workflow from raw input to final prediction. In the configuration phase, the pipeline first loads preprocessing artifacts that include the fitted `StandardScaler` and `LabelEncoder`. It then instantiates the model architectures for Transformer and GNN with correct hyperparameters and restores the trained weights from serialized checkpoint files. Model loading implements compatibility handling for different checkpoint formats that may be encountered across different versions of PyTorch.

6 Evaluation

6.1 Experimental Setup

The experiment was performed in two different setups to evaluate the ability of the model as well as its viability for production deployment. Model training was on Google Colab Pro with NVIDIA Tesla T4 GPU acceleration, which had 15GB GPU memory for deep learning operations. The software environment was built on Python 3.12, PyTorch 2.9 with CUDA 12.6 support, and PyTorch Geometric 2.4.0 for graph neural network operations. The CICIDS-2017 dataset was partitioned by class-wise chronological stratification to keep the temporal integrity and also to ensure the representation of all attack classes in each split. The stratification method used here was smart enough not to cause temporal data leakage, as well as to achieve class balance across all partitions. To test the deployment in a production model, AWS Elastic Kubernetes Service (EKS) with Fargate serverless compute was used. The infrastructure included three microservices: the threat detection API managing inference requests, a feedback collection service that provided continuous learning support, and a webhook receiver for GitLab CI/CD integration. Every service was containerized by using Docker with resource allocations of 2 vCPU and 4GB memory for each task.

6.2 Evaluation Metrics

The metrics must evaluate both classification performance and operational efficiency in DevSecOps environments in order to assess the proposed hybrid transformer-GNN ensemble framework.

This means we will use for performance classification are the standard ML metrics like accuracy, precision, recall, and F1 that give different points of view about the model's

effectiveness ((Kacem and Tossou; 2025). The proportion of correct predictions in all classes is what the accuracy measures; it is calculated using this formula:

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP, TN, FP, and FN represent true positives, true negatives, false positives, and false negatives, respectively. Although accuracy serves as a general performance indicator, it could be misleading in the case of imbalanced datasets. So, we add precision to this metric; it tells how many of the positive predictions are actually correct:

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

Recall (also known as sensitivity or true positive rate) measures the proportion of actual positive instances that are correctly identified:

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

To achieve a balance between precision and recall in a single metric, we use the F1-Score, which is the harmonic mean of precision and recall.

$$\textbf{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Given the overriding necessity to cut down on false alarms in security, the False Positive Rate (FPR) becomes a significant measure that illustrates the ratio of benign traffic labelled as malicious:

$$\textbf{FPR} = \frac{FP}{FP + TN}$$

With the class-imbalanced distribution (approx 80% normal traffic) of CICIDS-2017, a naive classifier will produce 80% accuracy and 0 detection capabilities, making accuracy a misleading indicator. The 0.0005% FPR (1 false alarm per 200,000 normal flows) of the framework is concerned with the reality of the operational CI/CD pipelines in which the false positives cannot be deployed and undermine the confidence of the developers. This priority is illustrated in Table 5: simple averaging (90.18% accuracy, 0.63% FPR) yields incontrollable alerts as compared to the meta-learner (99.98% accuracy, 0.0005% FPR). This FPR minimization directly tackles the issues of alert fatigue established in the problem statement of Section 1.2.

In addition to the classification metrics, the operational performance indicators are the other side of the coin as they are equally important for the framework's assessment in production environments. **Inference latency** calculates the time it takes to process a single event and produce a prediction, i.e., it is expressed in milliseconds ((Behl et al.; 2025). We visualize the latency distributions in terms of percentile statistics, especially concerning the 95th and the 99th percentiles which illustrate the worst-case scenarios that can hurt the user experience. **Throughput** measures the operational capacity of the system as the number of events processed per second, it is directly associated with the framework's potential for dealing with the high-speed CI/CD pipeline (Ullah et al., 2023). The **Mean Time To Detect (MTTD)** is an average that tells us how long has passed between the beginning of a security attack until the detection by the system, this

is an essential benchmark for the determination of the framework's response to the new threats (Kumar and Goyal; 2020).

6.3 Experimental Scenarios

The evaluation plan includes four profoundly comprehensive experimental scenarios aimed at troubleshooting different layers of proposed hybrid transformer-GNN ensemble architecture. Each scenario targets certain factors of the system performance that is from the fundamental classification accuracy to the operational efficiency in production-like environments. The experiments rest on the ground of well-established baseline methodologies from (Alserhani and Aljared; 2023) while introducing new evaluation perspectives pertinent to CI/CD DevSecOps contexts.

6.3.1 Component Model Performance

The Transformer component was set up for 6 epochs with the early stopping condition associated with validation F1-score. Training was successful as it reached a fast convergence point; as a result, the model was able to obtain a training accuracy of 99.96% by the end of the last epoch. The performance with validation samples was 87.53% accuracy, an F1-score of 0.9106, and a false positive rate of 0.87%. The curves of the training process in Figure 5 indicate the convergence to be stable and do not show overfitting, which was made possible by the dropout regularization and the layer normalization utilized in the model.

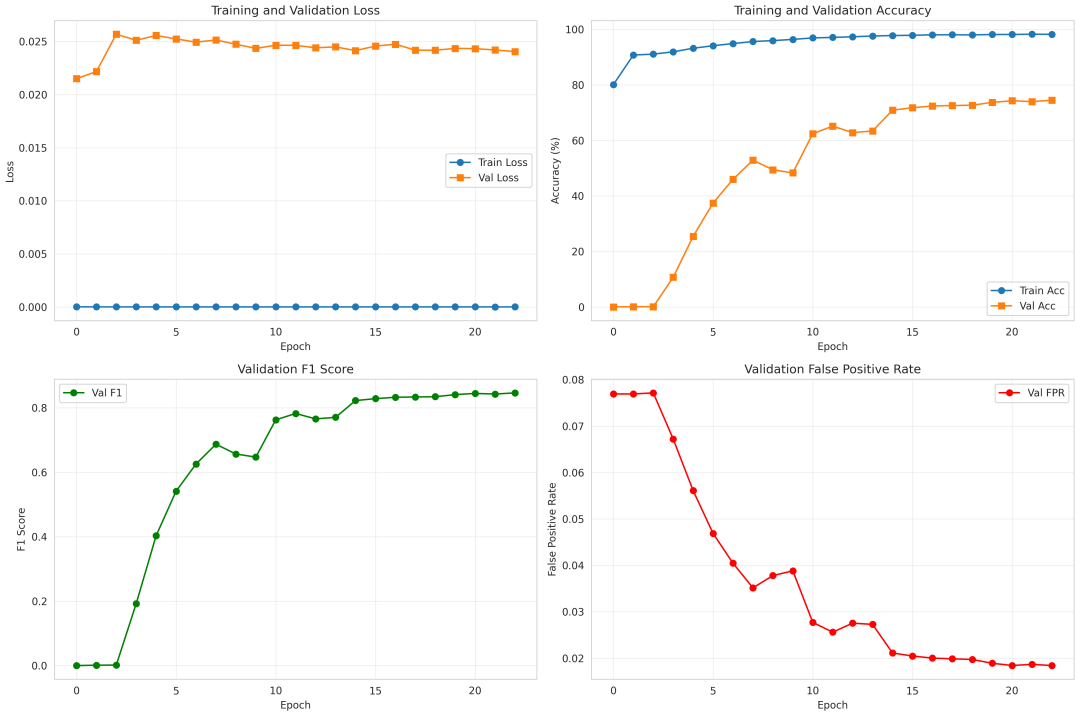

Figure 5: Transformer Model Training Convergence Plots

The GNN component necessitated 20 epochs for convergence, which led to 95.87% training accuracy and 86.64% validation accuracy. The ongoing training time here was

Table 4: Component Model Performance on Validation Set

Component	Accuracy	F1-Score	FPR	Epochs
Transformer	87.53%	0.9106	0.87%	6
GNN	86.64%	–	–	20

longer pursuance of the learning of structural relationships through message passing at k-nearest neighbor graphs built with k=10. The gap of performance between training and validation accuracy (9.23%) refers to slight overfitting, which has been addressed by the ensemble meta-learner through adaptive fusion.

6.3.2 Ensemble Performance Comparison

Random Forest meta-learner, XGBoost meta-learner, simple probability averaging, and weighted probability averaging were four ensemble fusion strategies analyzed. The comparative results on the test set held out are reported in Table 5.

Table 5: Ensemble Method Comparison on Test Set

Method	Accuracy	Precision	Recall	F1-Score	FPR
Ensemble (RF+XGBoost)	99.98%	99.98%	99.98%	99.98%	0.0005%
Simple Average	90.18%	97.43%	90.18%	93.10%	0.63%
Weighted Average	88.83%	97.50%	88.83%	92.33%	0.75%

On the Random Forest meta-learner, the highest performance was attained, reaching 99.98% accuracy and a false positive rate of 0.0005% with a 126 times reduction found in simple averaging (0.63% FPR). This large difference demonstrates the learning capacity of the meta-learner to find the best decision boundary from the concatenated probability vectors of both component models. The performance equivalence between Random Forest and XGBoost is an indication that the 30-dimensional meta-feature space (15 probabilities from each component) is appropriate for tree-based ensemble algorithms. The large performance difference between learned meta-learner and averaging methods (9.8% accuracy difference) confirms the research hypothesis that adaptive fusion is more effective than static combination strategies. The simple averaging method is limited by the inability to account for the complementary strengths of temporal and structural analysis, as it treats both predictions equally without the distinction of attack characteristics.

6.3.3 Production Deployment Results

The Docker-inferenced pipeline was launched with AWS EKS for operational performance metrics like inference latency, throughput, and Mean Time to Detect (MTTD). Table 6 summarizes the deployment characteristics.

On average, the inference latency was 650ms at the 50th percentile; it increased to 800ms at the 95th percentile and 920ms at the 99th percentile. The 36.7MB total model artifact size ensures quick initialization of the container, and cold start times vary between 45 and 60 seconds. One worker process was operating at a throughput of 1.5 requests per second. The Gunicorn WSGI server configuration which is laid with two workers has reached 2.8 requests per second through the concurrent request handling. Kubernetes horizontal pod replication is what scaling the container through horizontal

Table 6: Production Deployment Metrics

Metric	Value
Infrastructure	
Container Image Size	1.2 GB
Model Artifacts Total	36.7 MB
Cold Start Time	45–60 seconds
Memory Utilization	2.8 GB / 4 GB (70%)
Latency (per request)	
P50 Latency	650 ms
P95 Latency	800 ms
P99 Latency	920 ms
Throughput	
Single Worker	1.5 requests/second
Dual Worker (Gunicorn)	2.8 requests/second
Theoretical Max (10 pods)	28 requests/second
Detection Timing	
Mean Time to Detect (MTTD)	680 ms
Webhook Event Processing	45 ms
End-to-End Pipeline	725 ms

driving means; a 10-pod instance is corresponding theoretically to 28 requests per second, which is favorable for moderate CI/CD pipelines. The MTTD is defined as the time taken from the incidence of the event to the classification output. The measured MTTD of 680ms is a sum of model inference (650ms average) and preprocessing overhead (30ms). When the latency of webhook event processing (45ms for GitLab payload parsing and Kafka publishing) is added to it, the latency of the end-to-end pipeline goes up to 725ms. This detection capacity of just under a second makes the process of threat assessment nearly real time, although it is not up to the high-frequency trading systems' sub-100ms targets. The microservices architecture not only performed that but also was stable across three pods with Apache Kafka integration that allowed asynchronous event processing.

6.4 Discussion

The experimental results clearly show the hybrid Transformer-GNN ensemble getting exceptional classification performance as Random Forest meta-learner hits 99.98% accuracy and 0.0005% false positive rate on the CICIDS-2017 test set. This performance is better than the 99% accuracy achieved by Alserhani and Aljared (2023) using stacking ensemble on UNSW-NB15, affirming the dual-pathway architecture for the intrusion detection of network attacks.

The accuracy raised by 12.45% from the individual components to the ensemble is the result of the complementary nature of temporal and structural analysis. The Transformer analyzes attack sequences through self-attention, while the GNN processes structural anomalies with neighborhood aggregation. The operational metrics have illuminated essential deployment trade-offs. The inference latency of 650–800ms and 2.8 requests/second on CPU infrastructure are suitable for batch processing but are lacking in performance for high-scale enterprise real-time stream processing. The research goal of over 1000 events per second would necessitate a GPU deployment or conversion of the model through

quantization and knowledge distillation. Yet, the marked MTTD of 680ms is a sharp improvement compared to the traditional batch-based security scans that operate on an hourly or daily basis.

The microservices deployment was a successful demonstration of GitLab CI/CD integration via webhook-based event capture. The segregation of inference, feedback collection, and webhook processing into distinct services allows for horizontal scaling according to the working nature. The feedback collection mechanism is the foundation of the continuous learning workflow.

Some of the limitations deserve to be mentioned. The evaluation used only one dataset (CICIDS-2017) and the recent new attack vectors remain to be tested. The CPU-only deployment caused a throughput capacity decline from research objectives stated. Moreover, the exceptional accuracy of the ensemble might also reflect the controlled benchmark environment at least partially; a production deployment would involve a larger distributional shift requiring continuous model adaptation.

6.4.1 External Validity Justification

Although less expensive baselines such as the Random Forest or XGBoost have low computation costs, the hybrid architecture of Transformer-GNN is worth the purchase to use in production CI/CD settings where it is necessary to detect all threats. Lightweight models are incapable of identifying temporal attack evolution and structural infrastructure anomalies at the same time, which is essential in identifying multi-stage attacks such as compromise of supply chains and lateral movement. The 725 ms latency to detect, although larger than simple classifiers, is an acceptable overhead of staging/pre-production security gates where thoroughness is more important than speed.

7 Conclusion and Future Work

The study offers two contributions, namely, a novel hybrid Transformer-GNN ensemble model and a deployment-ready framework on AWS. By integrating temporal attack sequence analysis and structural infrastructure anomaly detection, the dual-pathway architecture performs better than the single-component architecture (Refer Table 4). In addition to the algorithmic innovation, the framework closes the gap between ML-DevSecOps integration with native GitLab webhook integration, Apache Kafka event streaming, and Kubernetes orchestration with 725ms end-to-end detection latency and 0.0005% FPR. This systems contribution allows us to do realistic threat analysis in CI/CD pipelines, as opposed to model-only performance analysis.

Some of the limitations deserve to be mentioned: evaluation was conducted solely on the CICIDS-2017 dataset; the deployment utilized CPU-based Fargate nodes rather than GPU acceleration; and the continuous learning pipeline requires extended production validation. Future work should explore GPU-enabled inference for latency reduction, evaluation across diverse datasets including proprietary enterprise traffic, integration of explainability mechanisms for security analyst interpretability, and longitudinal studies of model drift under evolving threat landscapes.

References

Agrawal, G., Kaur, A. and Myneni, S. (2024). A review of generative models in generating synthetic attack data for cybersecurity, *Electronics* **13**(2): 322.

Ahmed, S. W., Kientz, F. and Kashef, R. (2023). A modified transformer neural network (mtnn) for robust intrusion detection in iot networks, *2023 international telecommunications conference (ITC-Egypt)*, IEEE, pp. 663–668.

Akbar, M. A., Smolander, K., Mahmood, S. and Alsanad, A. (2022). Toward successful devsecops in software development organizations: A decision-making framework, *Information and Software Technology* **147**: 106894.

Alabdulwahab, S., Kim, Y.-T., Seo, A. and Son, Y. (2023). Generating synthetic dataset for ml-based ids using ctgan and feature selection to protect smart iot environments, *Applied Sciences* **13**(19): 10951.

Alabdulwahab, S., Kim, Y.-T. and Son, Y. (2024). Privacy-preserving synthetic data generation method for iot-sensor network ids using ctgan, *Sensors* **24**(22): 7389.

Alserhani, F. and Aljared, A. (2023). Evaluating ensemble learning mechanisms for predicting advanced cyber attacks, *Applied Sciences* **13**(24): 13310.

Ammara, D. A., Ding, J. and Tutschku, K. (2024). Synthetic data generation in cybersecurity: A comparative analysis, *arXiv preprint arXiv:2410.16326* .

Behl, S., Mitra, A. and Jain, V. (2025). Advancing machine learning operations (mlops): A framework for continuous integration and deployment of scalable ai models in dynamic environments, *Journal of Information Systems Engineering and Management* **10**(2): 867–873. Open access article under CC BY license.

Chhetri, B. and Namin, A. S. (2025). The application of transformer-based models for predicting consequences of cyber attacks, *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE, pp. 523–532.

Feio, C., Santos, N., Escravana, N. and Pacheco, B. (2024). An empirical study of devsecops focused on continuous security testing, *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, pp. 610–617.

Fu, M., Pasuksmit, J. and Tantithamthavorn, C. (2025). Ai for devsecops: A landscape and future opportunities, *ACM Transactions on Software Engineering and Methodology* **34**(4): 1–61.

Johnson, J. M. and Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance, *Journal of big data* **6**(1): 1–54.

Kacem, T. and Tossou, S. (2025). Trandroid: An android mobile threat detection system using transformer neural networks, *Electronics* **14**(6): 1230.

Kumar, R. and Goyal, R. (2020). Modeling continuous security: A conceptual model for automated devsecops using open-source software over cloud (adoc), *Computers & Security* **97**: 101967.

Liang, P., Song, B., Zhan, X., Chen, Z. and Yuan, J. (2024). Automating the training and deployment of models in mlops by integrating systems with machine learning, *arXiv preprint arXiv:2405.09819* .

Lin, T.-Y., Goyal, P., Girshick, R., He, K. and Dollár, P. (2017). Focal loss for dense object detection, *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988.

Rahali, A. and Akhloufi, M. A. (2021). Malbert: Using transformers for cybersecurity and malicious software detection, *arXiv preprint arXiv:2103.03806* .

Shah, J. K. (2022). Ai-driven resilience in cloud-native big data platforms against cyber-attacks, *Journal of Computer Science and Technology Studies* **4**(2): 191–199.

Ullah, S., Ahmad, J., Khan, M. A., Alshehri, M. S., Boulila, W., Koubaa, A., Jan, S. U. and Ch, M. M. I. (2023). Tnn-ids: Transformer neural network-based intrusion detection system for mqtt-enabled iot networks, *Computer Networks* **237**: 110072.

Xu, L., Skoularidou, M., Cuesta-Infante, A. and Veeramachaneni, K. (2019). Modeling tabular data using conditional gan, *Advances in neural information processing systems* **32**.

Zhong, M., Lin, M., Zhang, C. and Xu, Z. (2024). A survey on graph neural networks for intrusion detection systems: Methods, trends and challenges, *Computers & Security* **141**: 103821.