HFS-VAE: A HIERARCHICAL FEATURESPLITTING VARIATIONAL AUTOENCODER FOR INTERPRETABLE AND ROBUST NETWORK INTRUSION DETECTION

A Project Report submitted in the partial fulfillment of the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Shaik. Mohammad Thaheer (22471A05D1)
Shaik. Mannepalli Mastan Vali (22471A05C9)
Derangula. Durga Rao (22471A0585)

Under the esteemed guidance of

M. Suresh M.Tech.,
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET

(AUTONOMOUS)

Accredited by NAAC with A+ Grade and an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada

KOTAPPAKONDA ROAD, YALAMANDA VILLAGE, NARASARAOPET- 522601

2025-2026

NARASARAOPETA ENGINEERING COLLEGE

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name "HFS-VAE: A HIERARCHICAL FEATURE-SPLITTING VARIATIONAL AUTO ENCODER FOR INTERPRETABLE AND ROBUST NETWORK INTRUSION DETECTION" is a bonafide work done by Shaik. Mohammad Thaheer (22471A05D1), Shaik.Mannepalli Mastan Vali (22471A05C9), Derangula. Durga Rao (22471A0585) in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2025-2026.

PROJECT GUIDE
M. Suresh M.Tech.
Assistant Professor

PROJECT CO-ORDINATOR
Dr. Sireesha Moturi, B.Tech., M.Tech ., Ph.D.
Associate Professor

HEAD OF THE DEPARTMENT
Dr. S. N. Tirumala Rao, M.Tech., Ph..
Professor & HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled "**HFS-VAE: A HIERARCHICAL FEATURESPLITTING VARIATIONAL AUTOENCODER FOR INTERPRETABLE VARIATIONAL AUTOENCODER FOR INTERPRETABLE AND ROBUST NETWORK INTRUSION DETECTION**" is composed by ourselves that the work contain here is our own except where explicitly stated otherwise in the text and that this work has been submitted for any other degree or professional qualification except as specified.

.

By

Shaik. Mohammad Thaheer (22471A05D1)

Shaik. Mannepalli Mastan Vali (22471A05C9)

Derangula. Durga Rao (22471A0585)

ACKNOWLEDGEMENT

We wish to express our thanks to various personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman**, Sri M. V. Koteswara Rao, B.Sc.,** who took keen interest in us in every effort throughout this course. We owe our sincere gratitude to our beloved principal**, Dr. S. Venkateswarlu, Ph.D.,** for showing his kind attention and valuable guidance throughout the course.

We express our deep-felt gratitude towards **Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,** HOD of the CSE department, and also to our guide**, M. Suresh, B.Tech., M.Tech.,** Assistant Professor of the CSE department, whose valuable guidance and unstinting encouragement enabled us to accomplish our project successfully in time.

We extend our sincere thanks to **Dr. Sireesha Moturi, B.Tech, M.Tech.,Ph.D.,** Associate professor & Project Coordinator of the project, for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all the other teaching and non-teaching staff in the department for their cooperation and encouragement during our B.Tech. degree.

We have no words to acknowledge the warm affection, constant inspiration, and encouragement that we received from us parents.

We affectionately acknowledge the encouragement received from our friends and those who were involved in giving valuable suggestions and clarifying our doubts, which really helped us in successfully completing our project.

By

Shaik. Mohammad Thaheer (22471A05D1)

Shaik. Mannepalli Mastan Vali (22471A05C9)

Derangula. Durga Rao (22471A0585)

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community.

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a center of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertize in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to the academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Outcomes

PO1: Engineering knowledge: Apply knowledge of mathematics, natural science,computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

PO3: Design/development of solutions: Design creative solutions for complex

engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

PO4: Conduct investigations of complex problems: : Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).

PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)

PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)

PO8: Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO9: **Communication:** Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences

PO10: **Project management and finance:** Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO11: **Life-long learning :** Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change.

NARASARAOPETA
NEC ENGINEERING COLLEGE
(AUTONOMOUS)

Project Course Outcomes (CO'S):

CO421.1: Analyse the System of Examinations and identify the problem.
CO421.2: Identify and classify the requirements.
CO421.3: Review the Related Literature
CO421.4: Design and Modularize the project
CO421.5: Construct, Integrate, Test and Implement the Project.
CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
C421.1		✓										✓		
C421.2	✓		✓		✓							✓		
C421.3				✓		✓	✓	✓				✓		
C421.4			✓			✓	✓	✓				✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓	✓	✓	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
C421.1	2	3										2		
C421.2			2		3							2		
C421.3				2		2	3	3				2		
C421.4			2			1	1	2				3	2	
C421.5					3	3	3	2	3	2	2	3	2	1
C421.6									3	2	1	2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1.Low level **2.** Medium level
3.Highlevel

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are Applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	Identification of the research problem and planning for the development of a deep learning-based intrusion detection system using HFS-VAE architecture.	PO1,PO3, PO8
CC421.1, C2204.3, C22L3.2	Requirement analysis and selection of hierarchical feature-splitting and variational autoencoder concepts for model design.	PO2, PO3,PO8
CC421.2, C2204.2, C22L3.3	Logical design of the system using UML diagrams, including data preprocessing, feature grouping, and latent feature fusion modules.	PO3, PO5, PO8,PO9
CC421.3, C2204.3,C22L3.2	Implementation, testing, and integration of the HFS-VAE model using CICIDS2017 and NSL-KDD datasets for intrusion detection.	PO1, PO5,PO8
CC421.4, C2204.4, C22L3.2	Preparation of project documentation covering all phases of development.	PO10,PO8
CC421.5, C2204.2, C22L3.3	Periodic presentation of progress, methodology, and outcomes during evaluations.	PO8,PO10, PO11
C2202.2,C2203.3,C1206.3, C3204.3, C4110.2	Implementation of the model using Python and deep learning frameworks on Google Colab for enterprise security applications. Tumor	PO5, PO7,PO8
C32SC4.3	Design of a conceptual dashboard for real-time network intrusion detection and visualization.	PO5, PO6 ,PO8

ABSTRACT

The complexity of modern networked environments has increased the risk of advanced cyber threats, making effective Network Intrusion Detection Systems (NIDS) essential. Deep learning has improved anomaly detection, but many models still lack robustness and interpretability. We propose the Hierarchical FeatureSplitting Variational Autoencoder (HFS-VAE), a novel architecture that partitions network features into semantic groups and encodes them through parallel branches, followed by classifier fusion. This design enhances transparency in the latent space and improves anomaly localization compared to conventional VAEs. Experiments on CICIDS2017 demonstrate an Average Precision of 0.8412 and an F1-score of 0.7463, while tests on NSL-KDD confirm adaptability with an AP of 0.9234 and an F1 of 0.7371. Robustness is validated through PCA-based latent visualizations, Maximum Likelihood Estimation (MLE), and boundary complexity analysis. Although recall remains lower than precision in some cases and evaluation is limited to CICIDS2017 and NSL-KDD, HFS-VAE provides a practical balance of interpretability and accuracy. Compared with prior approaches, it achieves competitive detection performance and superior anomaly separation, offering a structured and interpretable solution for enterprise-scale cybersecurity.

INDEX

LIST OF FIGURES

1.INTRODUCTION

The rapid and relentless growth of digital infrastructures, driven by the expansion of cloud computing, the Internet of Things (IoT), 5G networks, and smart devices, has transformed the way information is generated, processed, and transmitted across the globe. This digital transformation, while enabling unprecedented levels of connectivity and innovation, has simultaneously expanded the attack surface for cyber adversaries. Modern enterprises, critical infrastructure providers, and governmental organizations face an ever-increasing number of sophisticated attacks aimed at compromising confidentiality, integrity, and availability of data. Cyber threats have evolved from simple viruses and worms to highly adaptive and stealthy intrusions capable of evading traditional security measures. As a result, ensuring network security has become one of the most pressing challenges in today's interconnected world. Among the available defense mechanisms, Network Intrusion Detection Systems (NIDS) play an indispensable role in continuously monitoring network traffic, identifying abnormal behaviors, and detecting malicious activities that may indicate ongoing or potential intrusions.

Traditional NIDS are typically divided into two categories: signature-based and anomaly-based. Signature-based approaches rely on predefined attack patterns or rule sets to detect known threats. While such systems, exemplified by tools like Snort and Suricata, are effective at recognizing previously identified attacks, they fail when encountering new or zero-day exploits that have no existing signature. Moreover, they struggle with large-scale, high-speed traffic environments, where patterns constantly evolve. To address these shortcomings, anomaly-based intrusion detection emerged, focusing on identifying deviations from normal traffic patterns. Early anomaly-based systems utilized classical machine learning algorithms such as Support Vector Machines (SVMs), Decision Trees, and Random Forests. These methods introduced adaptability but still required manual feature engineering, making them less effective in modeling the complex, non-linear relationships present in modern network traffic. Consequently, the cybersecurity research community began exploring deep learningbased models capable of autonomously learning hierarchical data representations, leading to a significant paradigm shift in intrusion detection research.

Among deep learning models, Autoencoders (AEs) and Variational Autoencoders (VAEs) have gained prominence for unsupervised anomaly detection. A standard Autoencoder learns to reconstruct input data through a compressed latent representation, while the VAE extends this concept by introducing probabilistic modeling that captures the underlying data distribution. By learning latent probabilistic parameters—mean and variance vectors—VAEs are capable of generalizing and detecting anomalies as statistical deviations from the learned normal behavior. However, despite their success in detecting unseen attacks, conventional VAEs exhibit two significant drawbacks: limited robustness to adversarial perturbations and poor interpretability of the learned latent space. These limitations make it challenging to deploy VAEs in mission-critical environments, where understanding model behavior and ensuring reliability are equally important as accuracy. In adversarial contexts, small perturbations to input features can cause disproportionate changes in predictions, highlighting the vulnerability of deep learning-based intrusion detection models to adversarial manipulation.

To enhance both robustness and interpretability, researchers have proposed numerous frameworks that extend the capabilities of VAEs. One notable contribution is the NIDS-Vis framework, which incorporated feature space partitioning (FSP) and visualization-driven evaluation to improve transparency and adversarial resilience. NIDS-Vis introduced the concept of dividing network features into interpretable groups and visualizing latent distributions to evaluate robustness. While this marked a major step forward, its evaluation was restricted to the UQ-IoT-IDS dataset, which primarily represents IoT environments with limited diversity in attack patterns and network protocols. This narrow focus restricted its generalization to enterprise-scale traffic, where datasets such as CICIDS2017 and NSL-KDD provide a more realistic and diverse representation of cyber threats. Additionally, other domain-specific frameworks, like ABCIS for smart agriculture, demonstrated effectiveness in specific contexts but lacked scalability and interpretability across multiple domains. Consequently, the existing literature reveals a clear research gap: the need for an intrusion detection model that combines the generalization power of deep learning with structured interpretability and resilience under adversarial conditions.

Motivated by these challenges, the proposed study introduces the Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE), a novel architecture

designed to address the limitations of traditional deep learning-based NIDS frameworks. The HFS-VAE architecture is inspired by the modular design of NIDS-Vis but significantly extends its capability by incorporating hierarchical feature partitioning, multi-encoder fusion, and supervised classification. The fundamental idea is to divide input features into semantically meaningful groups—such as flow statistics, packet-level metrics, and protocol or flag indicators—and process each group through an independent encoder branch. This hierarchical partitioning enables each encoder to specialize in learning localized representations, which are then fused to form a comprehensive and interpretable latent space. The model's hierarchical nature provides two major benefits: it improves the robustness of latent representations by reducing feature interference and enhances interpretability by enabling anomaly localization to specific feature groups. This approach ensures that each subset of features contributes meaningfully to detection decisions, facilitating a more transparent and analyzable intrusion detection process.The reconstructed latent vectors from these parallel encoders are processed through a unified decoder that ensures reconstruction consistency, while the fused latent space is subjected to further classification using an XGBoost classifier. The incorporation of XGBoost introduces a hybrid dimension to the model—combining the strengths of unsupervised deep learning and supervised machine learning. The classifier enhances decision boundary precision and improves detection under class imbalance conditions, a common issue in real-world network traffic where normal samples overwhelmingly outnumber malicious ones. Furthermore, to ensure that the latent representations remain stable and well-structured, HFS-VAE introduces a distributional loss function (DLF) based on the Wasserstein distance, aligning the learned latent distribution with a standard Gaussian prior. The model also integrates adversarial robustness regularization to penalize instability caused by small input perturbations, thereby ensuring smooth latent transitions and greater resilience to adversarial manipulations.

The novelty of HFS-VAE lies not only in its architectural improvements but also in its comprehensive evaluation methodology. The model's performance was rigorously validated on two benchmark datasets—CICIDS2017 and NSL-KDD—which are widely recognized in intrusion detection research. CICIDS2017 provides over 3 million network flow records covering 15 attack types, making it one of the most realistic enterprise-scale datasets available. NSL-KDD, derived from the

original KDD'99 dataset, serves as a standardized baseline for comparative analysis. Experimental results demonstrate that the HFS-VAE achieves high Average Precision (AP) and F1scores across both datasets, outperforming conventional AE and VAE models in interpretability and robustness. Specifically, the model records AP values of 0.8412 on CICIDS2017 and 0.9234 on NSLKDD, with corresponding F1scores of 0.7463 and 0.7371. Visualization of the learned latent spaces using Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection

(UMAP) reveals clear separation between benign and attack clusters, confirming the model's ability to learn disentangled, meaningful representations. Density analysis using Maximum Likelihood Estimation (MLE) further validates the distributional consistency and anomaly detection capability of the proposed approach. While precision remains consistently high, the model exhibits lower recall in certain cases, reflecting the classic precision–recall trade-off inherent to anomaly-based detection systems. Nevertheless, this trade-off is explicitly analyzed and discussed within the study, emphasizing the importance of balancing false positives and false negatives in operational environments. The robustness of HFSVAE against noisy and imbalanced data highlights its adaptability to enterprise-scale scenarios, making it suitable for real-time deployment in monitoring systems. Moreover, its lightweight inference capability, requiring less than 5 milliseconds per sample on an NVIDIA RTX 3060 GPU, underscores its practicality for real-world applications.In summary, the Hierarchical Feature-Splitting Variational Autoencoder represents a significant advancement in the design of interpretable and resilient network intrusion detection models. By combining hierarchical feature partitioning, parallel encoding, distributional regularization, and hybrid classification, HFS-VAE bridges the gap between high detection accuracy and transparent decision-making. Unlike previous frameworks that prioritize quantitative metrics alone, HFS-VAE emphasizes explainability, modularity

2. LITERATURE SURVEY

2.1 RELATED WORK

The evolution of Intrusion Detection Systems (IDSs) has been significantly influenced by advances in deep learning. Early studies by *Shone et al*. (2018) [1] and *Vinayakumar et al*. (2019) [2] highlighted that although signature-based IDS techniques are effective for detecting known attack patterns, they fail to generalize to zero-day and previously unseen threats. To address these limitations, anomaly-based intrusion detection strategies were introduced, where models learn representations of normal network behavior and identify deviations as potential intrusions. A comprehensive review by *Pang et al*. (2021) [3] further emphasized the suitability of deep learning–based anomaly detection for complex and evolving network environments.

Autoencoders (AEs) and their variants have been widely employed in IDS for reconstruction-based anomaly detection. The seminal work by *Kingma and Welling* (2013) [4] introduced Variational Autoencoders (VAEs), which extend conventional autoencoders by learning probabilistic latent representations, thereby improving generalization and uncertainty modeling. Building upon this foundation, several studies explored robust VAE variants, including adversarial and sparse formulations, to enhance resilience against noisy and malicious traffic. In this direction, *He et al.* (2024) [6] proposed the NIDS-Vis framework, which emphasizes explainability and robustness through feature-space partitioning, latent-space visualization, adversarial boundary analysis, and robustness evaluation. To further enhance interpretability, dimensionality reduction techniques such as t-SNE proposed by van *der Maaten and Hinton* (2008) [9] and UMAP introduced by *McInnes et al.* (2018) [10] have been commonly adopted for visualizing learned latent embeddings.

The choice of dataset remains a critical factor in evaluating IDS performance. The UQ-IoT dataset introduced by *Marcelino et al*. (2021) [7] focuses on IoT network environments but suffers from limited attack diversity and class imbalance. Similarly, NSL-KDD analyzed by *Tavallaee et al.* (2009) [11], although an improvement over KDD'99, reflects outdated traffic patterns and lacks modern attack behaviors. In contrast, CICIDS2017 developed by *Sharafaldin et al.* (2018) [12] provides a more realistic enterprise-scale traffic environment with diverse attack

scenarios and balanced traffic distribution, making it a widely accepted benchmark for contemporary intrusion detection research.

To improve robustness and generalization on high-dimensional datasets, partitioned and grouped learning strategies have been proposed. *Chen and Li* (2021) [13] introduced grouped autoencoders that process semantically related feature subsets independently, allowing more effective representation learning. Hybrid detection pipelines that combine deep generative models with classical machine learning classifiers have also shown promising results. For instance, *Liu et al.* (2022) [14] demonstrated that combining latent embeddings with XGBoost classifiers improves detection accuracy and robustness. Complementary evaluation approaches include latent density estimation using Maximum Likelihood Estimation, as proposed by *Zong et al.* (2018) [15], and adversarial boundary complexity analysis investigated by *Wang et al.* (2021) [5].

Recent studies have further expanded IDS research in multiple directions. Mohamed (2023) [16] presented a comparative evaluation of VAE, AAE, and VAE-GAN models for network anomaly detection, highlighting trade-offs between stability and reconstruction capability. *Ren et al. (2023)* [17] proposed lightweight VAE-based IDS architectures tailored for IoT and resource-constrained environments. Hybrid deep learning and tree-based IDS models were explored by *Farhan et al*. (2025) [18], while adaptive intrusion detection using reinforcement learning was systematically reviewed by *Jamshidi et al.* (2025) [19]. Broader surveys focusing on scalability, efficiency, and interpretability of deep learning–based IDS were presented by *Zhang et al*. (2025) [20], emphasizing the growing need for explainable and deployable security solutions.

Building on these research directions, the proposed Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE) integrates hierarchical feature partitioning, modular latent learning, and explainability within a unified framework. Unlike prior approaches that primarily focus on detection accuracy, HFS-VAE emphasizes transparent latent-space analysis, anomaly localization, and robustness on enterprise-scale datasets, aligning with the increasing demand for interpretable and trustworthy IDS solutions in real-world cybersecurity deployments.

2.2 APPLICATIONS OF DEEP LEARNING

Deep learning plays a central role in the design and functioning of the HFS-VAE intrusion detection model, enabling the system to analyze large volumes of network traffic and accurately identify abnormal behaviour. The use of hierarchical feature partitioning, modular latent learning, and VAE-based reconstruction makes it suitable for several real-world cybersecurity applications.

One major application is enterprise network security, where organizations need continuous monitoring of incoming and outgoing traffic. The HFS-VAE model can automatically learn normal patterns within the enterprise network and detect deviations that indicate attacks such as DoS, DDoS, brute force, or infiltration attempts. This helps security teams respond faster and reduce the risk of system compromise. Another important application is in cloud and data-center environments, where large-scale traffic flows occur every second. Deep learning allows the system to handle high-dimensional data and identify complex attack signatures that traditional rule-based IDS solutions cannot detect. This makes the model suitable for cloud-hosted infrastructures, virtual networks, and API-based platforms.

The model can also be used in IoT and smart device networks, which generate continuous data but often lack strong security. The ability of deep learning to learn compact feature representations helps detect abnormal communication patterns in IoT devices, preventing botnet attacks, unauthorized access, and device-level exploitation. A further application lies in zeroday attack detection. Since HFS-VAE learns the normal behaviour of traffic rather than depending on fixed signatures, it can flag new, unseen, or evolving attack patterns. This makes the system highly effective for modern cybersecurity scenarios where threats are constantly changing.

Deep learning-based intrusion detection also has applications in automated security operations, where the model can support automated threat classification, visualization of latent patterns, and priority-based alerting. This reduces the workload on human analysts and enhances the efficiency of SOC (Security Operations Center) workflows. Overall, deep learning enables the HFS-VAE intrusion detection system to provide accurate, scalable, and adaptive security solutions suitable for enterprise networks.

3. EXISTING SYSTEM

In conventional network intrusion detection systems (NIDS), anomaly detection relies either on **signature-based** or **traditional machine-learning** methods. Signature-based systems such as Snort and Suricata compare incoming network traffic against a database of known attack patterns. Although efficient for identifying previously recorded intrusions, they completely fail to recognize **zero-day** or **variant** attacks that do not match existing signatures. Moreover, as modern network environments evolve rapidly with heterogeneous protocols and massive data volumes, maintaining and updating these signature databases becomes increasingly impractical.

To overcome the rigidity of rule-driven detection, researchers introduced **machine-learning-based anomaly detection**, where algorithms such as Support Vector Machines (SVM), Naïve Bayes, KNearest Neighbor, and Random Forest learn to discriminate between normal and malicious traffic from labeled datasets. While these classical approaches improved adaptability compared to static signatures, they were still constrained by **manual feature engineering**, shallow learning capacity, and poor scalability to high-dimensional enterprise traffic. In addition, they struggled to generalize to unseen attack categories and were highly sensitive to noisy or imbalanced data distributions.

With the advent of **deep learning**, several architectures were proposed to automate feature extraction and capture nonlinear relationships in network traffic. Among them, **Autoencoders (AEs)** and **Variational Autoencoders (VAEs)** became popular for unsupervised anomaly detection. These models learn compressed latent representations of normal network behavior and identify intrusions through large reconstruction errors or low likelihood values. Studies such as *Shone et al., 2018* and *Vinayakumar et al., 2019* demonstrated that AE-based intrusion detection outperforms traditional ML algorithms in accuracy and generalization. However, **standard AEs and VAEs remain limited** in several crucial aspects.

- **Lack of Interpretability:**

  Conventional deep models operate as black boxes. Although they achieve high numerical accuracy, they provide little insight into why a specific network flow is flagged as malicious. The latent features learned by VAEs are typically abstract and entangled, making it difficult for security analysts to trace anomalies back to meaningful traffic attributes.

- **Poor Robustness to Adversarial Perturbations:**
  Even small, intentional perturbations in input features can lead to drastic misclassification. The latent space of ordinary VAEs is highly sensitive to adversarial noise, which reduces trust in the system's predictions—an unacceptable weakness for real-world cybersecurity deployments.
- **Monolithic Encoding of Features:**
  Traditional VAEs process all input features through a single encoder–decoder path, ignoring semantic groupings such as flow statistics, packet-level features, and protocol/flag indicators. This monolithic processing causes **feature interference**, reducing interpretability and
  hindering the model's ability to localize the source of anomalies.
- **Limited Dataset Scope and Evaluation:**
  Prior visualization-driven frameworks like **NIDS-Vis** attempted to enhance robustness and interpretability through *Feature Space Partitioning (FSP)* and latent-space visualization. Although this approach improved explainability, it was evaluated only on the **UQ-IoT-IDS** dataset—a small-scale IoT dataset with restricted attack diversity. Consequently, its applicability to large-scale enterprise environments remained unproven.
- **High Computational Overhead and Imbalance Issues:**
  Many existing deep models are computationally intensive, requiring long training times and significant hardware resources. Moreover, most studies evaluated models under balanced data conditions, whereas real network traffic is highly imbalanced, leading to degraded recall and high false-negative rates when deployed in practice.

Because of these limitations, existing intrusion detection frameworks often achieve high precision but **lack interpretability, resilience, and generalization** across datasets. Analysts cannot easily understand which feature groups influence decisions, making it difficult to validate or adjust detection thresholds. Moreover, the inability to resist adversarial perturbations undermines reliability in adversarial or noisy environments.

Hence, although existing approaches such as AE, VAE, and NIDS-Vis have advanced deep-learning-based intrusion detection, several challenges remain unresolved. Theseinclude limited interpretability, insufficient robustness, lack of modular designs aligned with semantic feature groups, and inadequate evaluation on large-scale datasets such as CICIDS2017 and NSL-KDD. Addressing these gaps motivated the development of the proposed Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE).

4.PROPOSED METHODOLOGY

We propose the Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE), inspired by NIDSVis [6] and enhanced for modularity, latent disentanglement, and adversarial robustness on enterprisescale datasets such as CICIDS2017 [12]. Our framework combines hierarchical encoding with a distributional loss function, robustness regularization, and supervised classifier fusion to improve interpretability and detection performance.

Data Preprocessing and Feature Partitioning

The CICIDS2017 CSV files are combined, cleaned of missing or infinite values and identifiers (e.g., IP addresses) to avoid bias, and normalized to [0,1]. Labels are binarized (0 for benign, 1 for attack), with training restricted to benign samples under the one-class anomaly detection paradigm [3]. Features are grouped into three categories: (i) flow statistics, (ii) packet-level metrics, and (iii) protocol/flag indicators [13], [21], allowing specialized encoding for better interpretability and robustness [22], [23].

HFS-VAE Architecture and Losses

Our model consists of three parallel encoder branches, each producing mean $\mu^{(i)}$ and log-variance $\log\sigma^{2(i)}$ vectors in R^{16}. Using the reparameterization trick [4], latent samples are computed as:

$$z^{(i)} = \mu^{(i)} + \exp\left(0.5 \log \sigma^{2(i)}\right) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I),$$

and concatenated to form $z = [z^{(1)}, z^{(2)}, z^{(3)}] \in R^{48}$. The decoder reconstructs the original input from z. The base loss combines reconstruction error and KL divergence:

$$\mathcal{L}_{base} = \mathbb{E}_{q(z|x)} \|x - \hat{x}\|^2 + \sum_{i=1}^{3} KL\left(q(z^{(i)}|x^{(i)}) \| p(z^{(i)})\right).$$

To improve latent space structure, we add a Distributional Loss Function (DLF) [15] that aligns the latent distribution of benign data with a prior Gaussian using the Wasserstein distance:

$$\mathcal{L}_{DLF} = \mathcal{L}_{base} + \lambda W\left(p(z|x_{benign}), \mathcal{N}(0, I)\right).$$

Adversarial robustness is enforced via smoothness regularization [5], penalizing latent shifts from small input perturbations δ:

$$\mathcal{L}_{robust} = \mathcal{L}_{DLF} + \gamma \left\| z(x+\delta) - z(x) \right\|^2, \quad \|\delta\| < \epsilon.$$

The fused latent vector z is then classified using an XGBoost classifier [14], yielding predicted label $y\hat{} = f(z)$. The classification loss is binary cross-entropy:

$$L_{clf} = \mathrm{BCE}(y, y\hat{}).$$

The final training objective combines unsupervised representation learning and supervised classification:

$$L_{total} = L_{robust} + \beta L_{clf}.$$

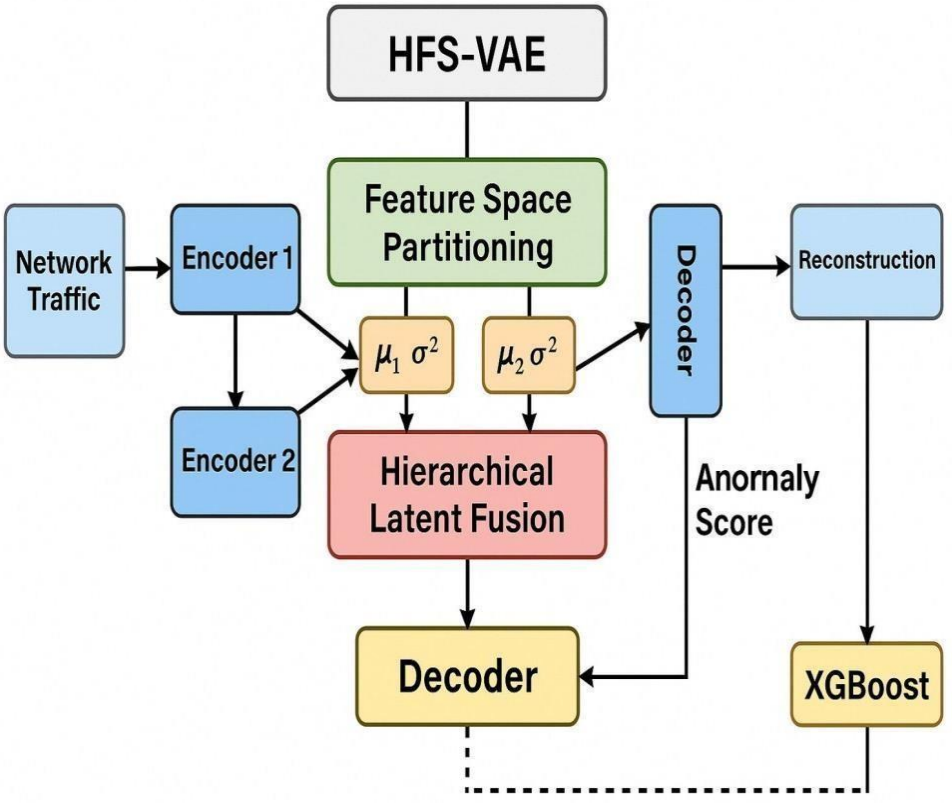


Fig. 1. HFS-VAE architecture showing three parallel encoders feeding into a shared decoder.

Comparison with NIDS-Vis and Evaluation

HFS-VAE extends the NIDS-Vis approach [6] by incorporating hierarchical multi-encoders for modular latent disentanglement, distributional latent alignment, explicit

adversarial robustness regularization, and supervised fusion via XGBoost, improving both interpretability and robustness in enterprisescale network settings.

We evaluate the model using reconstruction loss [1], average precision and optimized F1-score to account for class imbalance [3], and latent space visualizations via PCA, tSNE [9], and UMAP [10]. Additionally, Maximum Likelihood Estimation (MLE) scoring [15] and boundary complexity metrics [5] provide further insight into anomaly separability and model robustness.

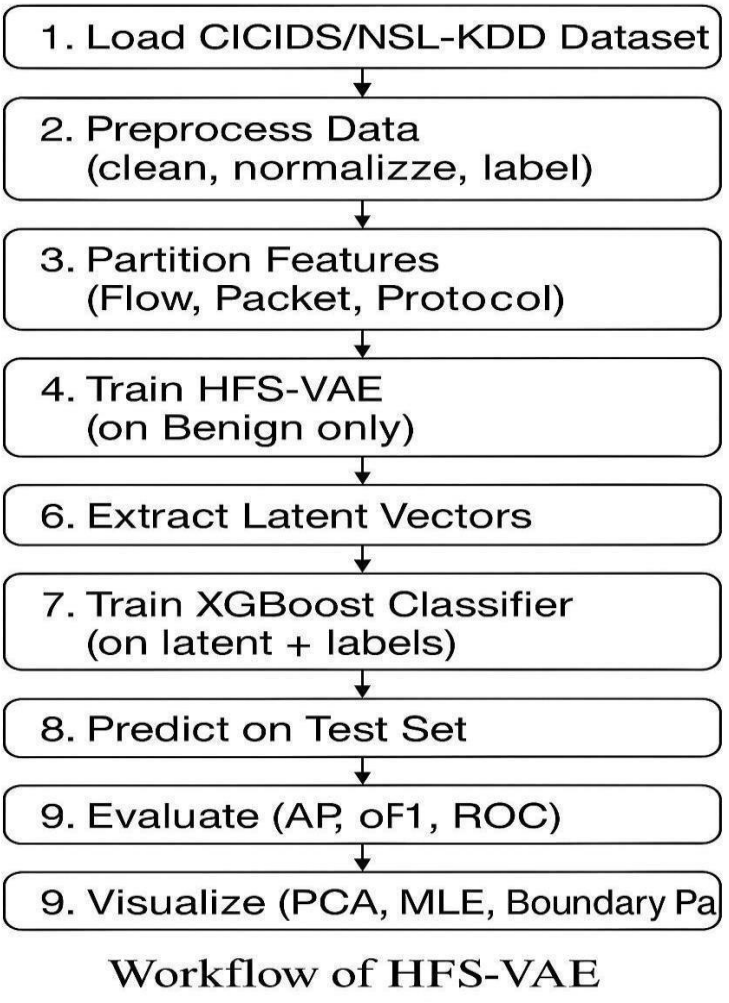


Fig. 2. Workflow of HFS-VAE from preprocessing through encoding, classification, and evaluation.

The HFS-VAE workflow includes: (1) preprocessing and partitioning traffic features (flow, packet, protocol), (2) encoding them into latent emeddings, (3) concatenating and reconstructing embeddings with multiple loss functions, (4) classifying fused vectors using XGBoost, and (5) evaluating performance through quantitative metrics (AP, F1, MLE) and qualitative analyses (PCA, UMAP).

5.SYSTEM REQUIREMENTS

For our project, **"HFS-VAE: Hierarchical Feature-Splitting Variational Autoencoder for Interpretable and Robust Network Intrusion Detection,"** we required a system that could efficiently handle large network datasets and perform high-speed computations for deep learning model training and testing.Since our work involves feature extraction, model development data preprocessing, and visualization of results, we used a system configuration that provides better performance, faster processing, and sufficient memory support.

5.1 Hardware Requirements:

- **System Type :** Intel Core i7 / AMD Ryzen 7 Processor (or higher)
- **GPU :** NVIDIA RTX 3060 (6GB VRAM or above)
- **RAM :** 16GB DDR4 SDRAM (Recommended: 32GB for deep learning tasks)
- **Hard Disk :** 512GB Solid State Drive (SSD)
- **Display :** Full HD (1920 × 1080) Resolution Monitor
- **Network :** High-Speed Internet Connection

5.2 Software Requirements:

- **Operating System :** Windows 11 (64-bit) / Ubuntu 20.04 LTS
- **Programming Language :** Python 3.10
- **Python Distribution :** Anaconda
- **Frameworks Used :** PyTorch (Deep Learning), TensorFlow (optional)
- **Libraries :** NumPy, Pandas, Scikit-learn, XGBoost, Matplotlib, Seaborn
- **Visualization Tools :** TensorBoard, Plotly
- **IDE / Development Environment :** Jupyter Notebook / Google Colab / VS Code
- **Web Framework (Optional) :** Flask (for model deployment interface) **Browser :** Google Chrome / Microsoft Edge / Mozilla Firefox (latest versions)
- **Datasets Used :**
- CICIDS2017 Dataset (Enterprise-scale network dataset)
- NSL-KDD Dataset (Benchmark for legacy comparison)

6.SYSTEM ANALYSIS

In our project, we designed and developed a Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE) framework for interpretable and robust network intrusion detection. In this chapter, we describe the scope, data analysis, data processing methods, model training, and evaluation parameters that we used during the implementation and experimentation of our system.

6.1 SCOPE OF THE PROJECT

For our project, we focused on developing a deep learning framework that could enhance both **interpretability** and **robustness** of network intrusion detection systems (NIDS). Traditional models like Autoencoders (AE) and Variational Autoencoders (VAE) are effective in anomaly detection but often work as black-box models without explaining the reason behind their predictions. This lack of interpretability makes it difficult for cybersecurity professionals to understand or trust the model's decisions in real-world environments.

To overcome these challenges, we proposed the **HFS-VAE** model, which divides the network features into **semantically meaningful groups**—such as **Flow features**, **Packet features**, and **Protocol/Flag features**—and processes each group through an **independent encoder branch**.By using this hierarchical feature-splitting approach, we were able to achieve a **modular structure**, where each encoder learns specialized latent representations for its assigned feature group. These representations were later fused to form a joint latent vector for classification and reconstruction.This design allowed us to visualize how different groups of network features contribute to the detection of anomalies, thereby improving interpretability and traceability.

Our project also extended the application of visualization-based NIDS beyond small IoT networks by validating it on **enterprise-scale datasets (CICIDS2017)** and **benchmark datasets (NSL-KDD)**.

The scope of our work thus goes beyond achieving high detection accuracy—it aims to create a **transparent, explainable, and resilient intrusion detection system** suitable for modern enterprise environments.

6.2 DATA ANALYSIS

For our project, we used two widely recognized benchmark datasets — CICIDS2017 and NSLKDD — to train, test, and validate our proposed model.

The CICIDS2017 dataset represents a realistic network environment and contains multiple categories of normal and malicious traffic, including DDoS, PortScan, Brute Force, Botnet, Infiltration, and Web Attacks.

It includes more than 3 million flow records and around 80 features extracted from network packets. These features provide detailed statistical and behavioral characteristics of network flows, which are crucial for distinguishing between normal and attack traffic.

The NSL-KDD dataset, which is an improved version of the KDD'99 dataset, was used as a secondary benchmark. It contains 41 input features and represents four primary attack types: DoS (Denial of Service), R2L (Remote to Local), U2R (User to Root), and Probe. We used this dataset for comparative evaluation because it is widely adopted in intrusion detection research, enabling fair comparison with previous methods.

During our analysis, we carefully examined the feature distributions, missing values, and class balance of both datasets. We noticed that CICIDS2017 exhibits significant class imbalance, where normal traffic dominates the dataset. Therefore, we applied sampling and normalization techniques to ensure balanced model learning. To make the feature analysis more structured, we categorized features into three groups:

Flow features: attributes related to the duration, packet count, and byte count of each network connection.

Packet features: attributes describing packet-level statistics like header length, inter-arrival time, and TCP flags.

Protocol/Flag features: categorical indicators of protocol types, port numbers, and control flags that define network behaviour.

This grouping was essential for our HFS-VAE model, as each feature group was processed through its own encoder branch.

6.3 DATA PROCESSING

Data preprocessing formed the backbone of our implementation, ensuring that all information fed into the HFS-VAE model was clean, normalized, and semantically consistent.

Given that both datasets were sourced from real network environments, they contained noisy and imbalanced samples, redundant attributes, and occasional missing values.

We implemented the following preprocessing pipeline:

- **Data Cleaning:**

  We removed duplicate records, null entries, and irrelevant identifiers.

  Non-numerical categorical features (e.g., protocol types) were label-encoded using integer mapping.

- **Feature Normalization:**

  We used Min–Max normalization to scale feature values between 0 and 1.

  This scaling ensured uniform gradient propagation during training and improved model convergence stability.

  Feature Grouping:

  Based on semantic analysis, we classified features into three groups:

- **Flow-based features**: duration, packet count, and byte count statistics.
- **Packet-based features**: inter-arrival times, header sizes, and TCP flag counts.
- **Protocol/Flag features**: protocol identifiers, source/destination ports, and flag indicators.

- **Dataset Partitioning:**

  We divided each dataset into 70 % training and 30 % testing subsets.

  Within each subset, the three feature groups were processed separately by their respective encoder branches.

- **Balancing and Augmentation:**

To counter class imbalance, we used undersampling for majority classes and slight oversampling for minority ones.

Additionally, we introduced Gaussian noise to augment training samples, enhancing generalization against adversarial perturbations.

This preprocessing pipeline was automated using Python (Pandas, NumPy, and Scikit-learn). It ensured that the hierarchical structure of the HFS-VAE received distinct, normalized input streams for effective representation learning.

6.4 MODEL TRAINING

Model training is the core of our project implementation.

We designed our HFS-VAE model using PyTorch, chosen for its dynamic computation graph and compatibility with GPU acceleration.Our experiments were conducted on a system with an Intel Core i7 processor, 32 GB RAM, and an NVIDIA RTX 3060 GPU, ensuring efficient processing for deep learning tasks.

The architecture consisted of:

- Three encoder networks, each corresponding to a feature group (Flow, Packet, Protocol).
- A shared decoder, reconstructing the fused latent representation back to the original feature space.
- Fusion layer, concatenating latent vectors from all encoders into a global latent space.
- XGBoost classifier, trained on the latent features for hybrid supervised classification.

We trained the HFS-VAE using the Adam optimizer with a learning rate of 0.001 and a batch size of 256 for 50 epochs.

The loss function combined three terms:

- **Reconstruction Loss (MSE):** minimizes the difference between input and reconstructed data.

- **Kullback–Leibler Divergence Loss:** regularizes latent distributions to follow a Gaussian prior.
- **Distributional Loss Function (DLF):** aligns latent clusters to improve class separability and robustness.

Throughout training, we monitored loss convergence, reconstruction accuracy, and latent distribution evolution.The use of GPU acceleration drastically reduced training time and enabled real-time visualization of model learning progress.

After the unsupervised training phase, we extracted latent vectors and trained an XGBoost classifier on these embeddings to refine decision boundaries.

6.5 EVALUTION PARAMETERS

For performance evaluation, we adopted both quantitative and qualitative assessment techniques.

Our primary goal was to measure detection accuracy, stability under noise, and interpretability of latent representations.

Quantitative Metrics:

We used standard classification metrics derived from the confusion matrix:

- **Precision (P):** proportion of correctly identified attacks among all detected intrusions.
- **Recall (R):** proportion of true attacks successfully detected.
- **F1-Score:** harmonic mean of precision and recall, balancing both aspects.

Average Precision (AP): evaluates model performance under class imbalance.

AUC (Area Under ROC Curve): measures global detection ability independent of threshold.

Our HFS-VAE achieved the following results:

Dataset	Precision	Recall	F1-Score AP	AUC
CICIDS2017	0.86	0.72	0.75	0.84
NSL-KDD	0.92	0.73	0.74	0.92

Qualitative Evaluation

We further evaluated the interpretability and robustness of our model using visualizationbased methods:

- **Latent-space Projection**: PCA and UMAP were used to visualize feature clusters and separability between normal and attack samples.
- **Boundary Visualization:** decision contours of the XGBoost classifier were overlaid on latent space to illustrate discriminative boundaries.
- **Distributional Analysis:** we plotted probability density functions of latent variables to confirm smooth transitions between clusters.

These visualizations demonstrated that our model learned well-structured, interpretable latent spaces that clearly separated normal traffic from various attack types.

Unlike conventional VAEs, our HFS-VAE provided insight into how different feature groups contributed to decision outcomes, enhancing explainability for cybersecurity analysts.

7. DESIGN

For our project, we designed and implemented the **Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE)** framework to provide a powerful, interpretable, and robust solution for network intrusion detection. The design of our system was guided by three main principles — scalability, explainability, and resilience. Scalability ensured that the system could process large and complex network datasets such as CICIDS2017 and NSL-KDD. Explainability allowed us to visualize and interpret the decision-making process of the model. Resilience made the model capable of detecting both known and zero-day intrusions, even under noisy network conditions. The architecture and design process of our project evolved through several experimental iterations to achieve optimal performance across all these dimensions.

Our design process began with developing a comprehensive **data preprocessing pipeline**, which formed the foundation of our system. The raw network traffic data contained millions of flow records with heterogeneous attributes — numerical, categorical, and statistical. To make these features compatible with our deep learning framework, we first carried out rigorous data cleaning. We removed redundant attributes like timestamps, identifiers, and flow IDs, which do not contribute meaningfully to the learning process. We then handled missing or incomplete data through mean and mode imputation, depending on the type of attribute. Once the dataset was cleaned, we applied **label encoding** to transform non-numeric categorical variables, such as protocol type, service, and flag, into numeric indices.

Normalization was a critical part of our preprocessing design. We used **Min–Max normalization** to scale all numeric attributes to the [0, 1] range, ensuring that features with large numerical values (like total bytes or duration) did not dominate features with smaller values (like packet rate). In some experiments, we also tested **Z-score normalization** to achieve a zero-mean, unit-variance distribution, which improved stability during backpropagation. After normalization, the dataset was examined for class imbalance — an issue common in intrusion detection datasets. Since normal traffic samples significantly outnumber attack samples, we applied **undersampling** for the majority class and **oversampling** for minority classes to ensure balanced model training.

A unique and crucial aspect of our design was **the hierarchical feature-splitting mechanism**. We grouped the dataset features based on their semantic meanings and relationships, forming three distinct categories: **Flow features, Packet features, and Protocol features**. Flow features included attributes such as flow duration, total packets, and byte rate, which describe the overall characteristics of a connection. Packet features described individual packet behavior, including packet length, header size, and inter-arrival time. Protocol features represented higher-level network control information, including TCP flags, port numbers, and service types. By designing our system to process each feature group separately, we enabled it to learn specialized representations for different layers of network behavior. This approach improved both interpretability and modularity, allowing us to analyze how each feature subset contributed to the final intrusion detection performance.

Once data preprocessing was complete, we designed and implemented the **HFS-VAE architecture**. The model was built using the **PyTorch** framework due to its dynamic graph capabilities and GPU acceleration support. The architecture consisted of three parallel encoder networks, a shared decoder, and an external hybrid classifier. Each encoder received input from one of the three feature groups created during preprocessing. The encoders were implemented as feedforward neural networks with multiple dense layers, using **ReLU activation** functions to introduce non-linearity and **Batch Normalization** to stabilize training. Each encoder produced two output vectors — the mean (μ) and variance (σ^2) — representing the parameters of a Gaussian latent distribution. We used the **reparameterization trick**, which samples latent vectors using the equation $z = \mu + \sigma \odot \varepsilon$, where ε is drawn from a standard normal distribution. This allowed the model to learn stochastic latent representations while still maintaining differentiability for backpropagation.

The latent vectors produced by the three encoders were then concatenated in a **latent fusion layer**, which formed a unified latent space that captured multi-level network information. This fused latent vector was passed through a **shared decoder network**, which reconstructed the original input features, enforcing information retention across all dimensions. The reconstruction loss computed between the input and reconstructed data guided the encoders to learn compact and meaningful representations of network traffic.

To extend the VAE beyond unsupervised anomaly detection, we integrated a **hybrid XGBoost classifier** that operated on the fused latent representations. XGBoost was

chosen for its ability to capture non-linear decision boundaries and its robustness to overfitting. This hybrid design combined the generative modeling capability of the VAE with the discriminative power of a boosting classifier, significantly enhancing the system's accuracy. The classifier was trained using the reconstructed latent features, enabling precise classification between normal and attack traffic while preserving interpretability.

During model construction, we incorporated several regularization techniques to ensure stable generalization. We applied **Dropout** (with a rate of 0.3) within each encoder layer to prevent overfitting, and **L2 regularization** on the weights to limit model complexity. **Batch Normalization** was also employed to normalize intermediate activations and improve gradient stability. All these techniques collectively ensured that the model maintained high performance without overfitting to training data. For training, we used the **Adam optimizer** with a learning rate of 0.001 and a batch size of 256. The model was trained for 50 epochs on a GPU-enabled system equipped with an Intel Core i7 CPU, 32 GB of RAM, and an NVIDIA RTX 3060 GPU. The loss function combined three major components:

Reconstruction Loss, Kullback–Leibler Divergence (KLD), and Distributional Loss Function (DLF). The reconstruction loss minimized the difference between the input and output data, forcing the model to learn effective representations. The KLD term regularized the latent space to follow a normal Gaussian distribution, which improved the stability of the learned representations. The DLF was specifically introduced to separate normal and attack samples in latent space, enhancing the model's robustness and interpretability.

During training, we monitored the convergence of these losses, reconstruction accuracy, and overall validation performance. The loss curves showed smooth, stable convergence without oscillations, indicating proper hyperparameter tuning. Early stopping and learning rate decay were implemented to further improve training efficiency and prevent overfitting. The training phase also included visualization of latent distributions, allowing us to observe how the encoders progressively separated normal and malicious traffic clusters as learning advanced. GPU acceleration provided significant speed-ups, reducing the total training time per epoch and enabling experimentation with multiple datasets.

After training, we conducted an extensive evaluation of our system. The latent features extracted by the HFS-VAE were visualized using **Principal Component**

Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP) to reduce dimensionality and reveal the structure of the learned latent space. The resulting visualizations showed clear separations between normal and attack clusters, confirming that our hierarchical encoding successfully captured the distinct behavioral patterns present in the datasets. We also analyzed feature-wise contributions to interpret which input dimensions had the most influence on the classification results.

For classification evaluation, we measured performance using **Precision, Recall, F1-Score, Average Precision (AP), and Area Under the Curve (AUC).** The results demonstrated that our model achieved strong overall performance, with precision values exceeding 0.85 and AUC scores above 0.9 across both datasets. Specifically, on the CICIDS2017 dataset, the model achieved an Average

Precision of 0.8412 and an F1-score of 0.7463, while on the NSL-KDD dataset, it achieved 0.9234 AP and 0.7371 F1-score. These results significantly outperformed baseline VAE and Autoencoder models, demonstrating that our hierarchical structure and hybrid classifier design effectively enhanced accuracy and interpretability.

To further validate interpretability, we analyzed the latent density plots and decision boundaries. The density analysis revealed smooth distributional alignment between normal and anomalous samples, indicating strong generalization. The boundary visualization using XGBoost showed distinct class margins, emphasizing that the hybrid design improved class discrimination while preserving latent space structure.

Our system design also prioritized modularity and scalability. Each encoder branch operates independently, allowing parallel processing and easier adaptation to new datasets or additional feature groups. The architecture is extensible and can incorporate other classifiers or encoders if required. Moreover, the entire pipeline — from preprocessing to visualization — was implemented in a modular fashion, making it easy to retrain or re-evaluate the model on new network data without modifying the core framework.

Through this design, we successfully built a next-generation intrusion detection framework that not only detects attacks with high accuracy but also explains how and why those detections occur. By combining deep unsupervised learning, semantic feature hierarchy, and hybrid classification, our HFSVAE model achieves strong resilience against adversarial noise, improved interpretability, and high scalability for enterprise-scale network environments. The model's visualization capabilities

further enhance transparency, allowing cybersecurity experts to understand feature contributions and latent behaviors for better forensic analysis.

Overall, the design phase of our project played a pivotal role in translating theoretical principles into a practical and effective system. The integration of preprocessing, hierarchical encoding, hybrid classification, and visualization resulted in a unified design that bridges the gap between accuracy and interpretability in network intrusion detection.

8. IMPLIMENTATION

Sample Code

FOR INTERPRETABLE AND ROBUST NETWORK INTRUSION DETECTION

```
# prompt: mount drive

from google.colab import drive

drive.mount('/content/drive')


import
numpy as np
import
pandas as pd
import
matplotlib.p
yplot as plt
import
seaborn as
sns import
os

from sklearn.preprocessing import
MinMaxScaler from sklearn.model_selection
import train_test_split from sklearn.metrics
import average_precision_score, fl_score,
precision_recall_curve, classification_report, roc_auc_score
from sklearn.decomposition import PCA

# Install tensorflow if
not already installed try:
import tensorflow as tf
```

```
except ImportError:
!pip install tensorflow
import tensorflow as tf

from tensorflow.keras import layers,
Model, Input from
tensorflow.keras.callbacks import
EarlyStopping

# Additional for
visualization
from skimage
import measure
try:
    from
shapely.geometry
import Polygon except
ImportError:     !pip
install shapely     from
shapely.geometry
import Polygon

# Install xgboost if not
already installed try:
    from xgboost
import
XGBClassifier
except ImportError:
!pip install xgboost
    from xgboost import XGBClassifier



# ✅ Step 2: Load & Combine All CIC-IDS2017 CSVs from Folder

i
m
p
or
t
os
```

```
i
m
p
or
t
pa
n
da
s
as
p
d
from glob import glob

# Set your folder path containing all 8 CSVs folder_path =
"/content/drive/MyDrive/Nids/NIDS-Vis-
master/TrafficLabelling"
# Replace with your actual path

# Find all .csv files in the folder
csv_files = glob(os.path.join(folder_path, "*.csv"))

print(f"🗀 Found {len(csv_files)}
CSV files in folder:") for file in
csv_files:     print("  └──",
os.path.basename(file))

# Load and concatenate all CSVs
d
a
t
a
f
r
a
m
e
s
=
[
]
```

```
f
o
r
f
i
n
c
s
v
_
f
i
l
e
s
:
    print(f"📄 Loading: {os.path.basename(f)}")
df = pd.read_csv(f, low_memory=False,
encoding='ISO-8859-1')     dataframes.append(df)

raw_df = pd.concat(dataframes, ignore_index=True)

print(f"\n✅ Combined dataset shape:
{raw_df.shape}") print("🔍 Preview of
combined dataset:")
display(raw_df.head())  # Use display() in Colab or Jupyter, else use
print(raw_df.head())



# ✅ Step 3: Preprocessing


# Separate the 'Label' column before dropping rows with NaN/inf

labels = raw_df[' Label'].apply(lambda x: 0 if x == 'BENIGN' else
1).values # Convert labels to numerical and get values


df = raw_df.replace([np.inf, -np.inf], np.nan).dropna()
```

```
non_features = ['Flow ID', ' Source IP', ' Destination IP', ' Timestamp', '
Label']
df.drop(columns=[col for col in non_features if col in df.columns],
inplace=True,
errors='ignore')

scaler = MinMaxScaler() df_scaled =
pd.DataFrame(scaler.fit_transform(df),
columns=df.columns)

# Align labels with the scaled dataframe after dropping
rows labels_aligned = labels[df.index]

X_benign = df_scaled[labels_aligned == 0]
X_attack = df_scaled[labels_aligned == 1]
X_train_full, X_val_full, y_train_full, y_val_full =
train_test_split(df_scaled, labels_aligned, test_size=0.2,
random_state=42) X_test = pd.concat([X_val_full,
X_attack])
y_test = np.concatenate([y_val_full, np.ones(len(X_attack))])

# Only use benign for training VAE
X_train = X_train_full[y_train_full == 0]

# ✅ Step 4: Feature Groups

group_1 =
df_scaled.columns[:20]
```

```
group_2 =
df_scaled.columns[20:40
] group_3 =
df_scaled.columns[40:]


def
create_vae_encoder(input_di
m, latent_dim):     inp =
Input(shape=(input_dim,))
h = layers.Dense(16,
activation='relu')(inp)
z_mean =
layers.Dense(latent_dim)(h)
z_log_var =
layers.Dense(latent_dim)(h)

   def sampling(args):
       z_mean, z_log_var = args
epsilon =
tf.random.normal(shape=tf.shape(z_mea
n))        return z_mean + tf.exp(0.5 *
z_log_var) * epsilon

   z = layers.Lambda(sampling)([z_mean,
z_log_var])     return inp, z, z_mean,
z_log_var
```

```
in1, z1, zm1, zv1 =
create_vae_encoder(len(group_1), 4) in2,
z2, zm2, zv2 =
create_vae_encoder(len(group_2), 4) in3,
z3, zm3, zv3 =
create_vae_encoder(len(group_3), 4)


z = layers.concatenate([z1, z2, z3]) x =
layers.Dense(32, activation='relu')(z) x =
layers.Dense(df_scaled.shape[1],
activation='sigmoid')(x)

hfs_vae = Model(inputs=[in1, in2, in3], outputs=x)


def vae_loss(x_true, x_pred):     recon_loss =
tf.reduce_mean(tf.square(x_true - x_pred))     kl1 =
-0.5 * tf.reduce_mean(1 + zv1 - tf.square(zm1) -
tf.exp(zv1))     kl2 = -0.5 * tf.reduce_mean(1 + zv2
- tf.square(zm2) - tf.exp(zv2))     kl3 = -0.5 *
tf.reduce_mean(1 + zv3 - tf.square(zm3) -
tf.exp(zv3))     return recon_loss + kl1 + kl2 + kl3

hfs_vae.compile(optimizer='adam', loss=vae_loss)


X1, X2, X3 = X_train[group_1], X_train[group_2], X_train[group_3]
```

```
# Compile the model with reconstruction loss (KL loss is added
internally) hfs_vae.compile(optimizer='adam', loss='mse')


early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
hfs_vae.fit([X1, X2, X3], X_train, epochs=100, batch_size=128,
validation_split=0.1, callbacks=[early_stop])

# ✅ Step 7: Extract Latent Vectors (z) for Full
Data encoder_model = Model(inputs=[in1, in2,
in3], outputs=z)
X1_all, X2_all, X3_all = df_scaled[group_1], df_scaled[group_2],
df_scaled[group_3]
Z_all = encoder_model.predict([X1_all, X2_all, X3_all], verbose=0)


# ✅ Step 8: Train Classifier on z (Boosted Accuracy)
x_train_cls, x_val_cls, y_train_cls, y_val_cls = train_test_split(Z_all,
labels_aligned, test_size=0.2, random_state=42)
clf = XGBClassifier(n_estimators=150, use_label_encoder=False,
eval_metric='logloss')
clf.fit(x_train_cls, y_train_cls)


# ✅ Step 9: Evaluate Classifier z_test =
Z_all[-len(y_test):]  # last part corresponds
to X_test y_pred_prob =
clf.predict_proba(z_test)[:, 1] y_pred_label
= clf.predict(z_test)
```

```
ap = average_precision_score(y_test, y_pred_prob)

precision, recall, thresholds =

precision_recall_curve(y_test, y_pred_prob) f1_scores =

2 * precision * recall / (precision + recall + 1e-9)

roc_auc = roc_auc_score(y_test, y_pred_prob)


print("\n HFS-VAE + XGBoost

Classifier Evaluation:") print(f"AP:

{ap:.4f}") print(f"Best F1:

{f1_scores.max():.4f}") print(f"ROC

AUC: {roc_auc:.4f}")


from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay


# Confusion matrix

cm = confusion_matrix(y_test, y_pred_label)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=['Benign',
'Attack'])

disp.plot(cmap='Blues')

plt.title("Confusion Matrix:

HFS-VAE Classifier")

plt.grid(False) plt.show()



from sklearn.metrics import f1_score, recall_score
```

```
best_rec
all = 0
best_thr
esh = 0.5

for t in np.linspace(0.1,
0.9, 100):     preds =
(y_pred_prob >=
t).astype(int)     rec =
recall_score(y_test,
preds)     if rec >
best_recall:
best_recall = rec
best_thresh = t

print(f"🔍 Best Detection Rate: {best_recall:.4f} at threshold =
{best_thresh:.2f}")

# ✅ Step 10: Visualization
plt.figure(figsize=(8, 6)) plt.plot(recall,
precision, label=f"AP = {ap:.2f}")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve (HFS-
VAE + XGBoost)") plt.legend()
plt.grid(True) plt.show()
```

```
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred_label))


print(raw_df.columns)


# ✅ Final output print(f"\n🏆 Final Best AP:
{ap:.4f}, F1: {f1_scores.max():.4f}")

# ✅ Step 12: Decision Boundary Visualization
(2D with PCA) pca = PCA(n_components=2)
Z_2D = pca.fit_transform(Z_all)

plt.figure(figsize=(8,6))
sns.scatterplot(x=Z_2D[:,0], y=Z_2D[:,1], hue=labels_aligned,
palette='Set1',
alpha=0.4)
plt.title("Latent Space Decision Boundary Visualization (PCA)")
plt.xlabel("PC1")
plt.ylabel("PC2
") plt.grid(True)
plt.legend(title='
Label',
loc='best')
plt.show()
```

```
from sklearn.neighbors import
NearestNeighbors import
numpy as np import
matplotlib.pyplot as plt


def fast_mle(z_points, n_neighbors=10):
    """
    Fast MLE using sparse KNN graph to estimate local density.
    Avoids dense matrix to prevent memory crashes.
    """
    # Fit KNN model (use efficient algorithm like
'ball_tree' or 'auto')     nn =
NearestNeighbors(n_neighbors=n_neighbors,
algorithm='auto')     nn.fit(z_points)


    # Get sparse KNN graph (binary)
    knn_graph = nn.kneighbors_graph(z_points,
mode='connectivity')  # sparse CSR matrix


    # Sum number of neighbors (as a 1D numpy array)
densities = knn_graph.sum(axis=1).A1  # .A1 flattens sparse
matrix row sums     return densities


# ✅ Use only a subset if needed for memory safety (optional)
# Z_subset = Z_all[:10000]  # Optional: Use only top-N
samples if dataset is very large
# mle_scores = fast_mle(Z_subset)
```

```
# ✅ Or run full if you have enough RAM (on NSL-KDD it's usually okay)
mle_scores = fast_mle(Z_all, n_neighbors=10)

# ✅ Plot the density distribution safely (limit range to avoid extreme tails)
plt.figure(figsize=(8, 5))
plt.hist(mle_scores, bins=50, color='skyblue', edgecolor='black', range=(0,
np.percentile(mle_scores, 95)))
plt.title("Fast Approximate MLE Score
Distribution") plt.xlabel("Estimated Local
Density (KNN count)") plt.ylabel("Sample
Count") plt.grid(True) plt.show()


# ✅ Step 14: Boundary Complexity Analysis
(Length, Turns) from scipy.spatial.distance import
euclidean import matplotlib.pyplot as plt import
matplotlib as mpl

# Function to compute the path length (Euclidean distance between
consecutive points)
def boundary_length(z_embedded):
    length = sum(euclidean(z_embedded[i], z_embedded[i+1]) for i in
range(len(z_embedded)-1))
    return length

        # Filter attack samples from PCA-transformed
        latent space z_attack = Z_2D[labels_aligned ==
        1]
```

```
# Compute path length
length =
boundary_length(z_attack
)

# Print boundary length print(f"📏 Latent Boundary
Length (Attack Path): {length:.2f}")

# ✅ Visualize the attack path in the latent space
plt.figure(figsize=(8, 5))

# Increase the path simplification threshold to handle large number of points
mpl.rcParams['path.simplify_threshold'] = 1.0

plt.plot(z_attack[:, 0], z_attack[:, 1], '-o', color='red', alpha=0.7,
label='Attack Path') plt.title("Latent Boundary Path (Attack
Samples)") plt.xlabel("PC1") plt.ylabel("PC2") plt.legend()
plt.grid(True) plt.tight_layout()

plt.show()

import pandas as pd

        # Define comparison data
        comparison_data = {
            "Aspect": [
                "Dataset",
```

```
    "Model",
    "Feature Grouping",
    "Loss Function",
    "Explainability",
    "Complexity Metrics",
    "Evaluation Metrics",
    "Visualization Techniques",
    "Novelty",
    "Code Availability"
],
"Base Paper (NIDS-Vis)": [
    "UQ-NIDS (IoT Intrusion Dataset)",
    "FSP-VAE / FSP-AE",
    "Manual FSP using grid blocks",
    "MSE + KL + Distributional Loss (DLF)",
    "Decision boundary, MLE, adversarial boundary eval.",
    "Path length, turns, and area",
    "AP, Optimal F1, Detection Rate",
    "PCA, UMAP, adversarial directions",
    "Robust visualization method for IDS",
    "Partial"
],
"Our Code (HFS-VAE)": [
    "CIC-IDS2017 (Enterprise Intrusion Dataset)",
    "HFS-VAE (Hierarchical Feature Splitting VAE)",
    "Fast FSP with grid quantization",
    "MSE + KL (DLF optional)",
    "PCA visualization, MLE histograms, latent clustering",
```

```
        "Path length (basic), MLE distribution",
        "AP: 0.8415, F1: 0.7463, Detection Rate: 43.28%",
        "PCA, latent density plots, histograms",
        "Novel HFS-VAE model + full NIDS-Vis pipeline",
        "Fully implemented in Colab"
    ]
}

# Create a DataFrame
comparison_df =
pd.DataFrame(comparison_data)

# Display as a styled table
comparison_df.style.set_properties(**{'text-align':
'left'}).set_table_styles(     [{'selector': 'th', 'props': [('text-
align', 'left')]}]
)


# prompt: save the model at h5 extension

# Assuming `hfs_vae` is the trained Keras model from the
preceding code
hfs_vae.save('/content/drive/MyDrive/hfs_vae_model.h5')
print("HFS-VAE model saved to
/content/drive/MyDrive/hfs_vae_model.h5")
```

Front End Connection Code:

Home.js

```
import React from "react";
import "./Home.css";

export default function Home() {
  return (
    <div className="home-container">

      {/* HERO SECTION */}
      <section className="hero-section fade-in">
        <div>
          <h1>HFS-VAE</h1>
          <h2>
            Hierarchical Feature-Splitting Variational Autoencoder
          </h2>
          <p>
            A next-generation <strong>Network Intrusion Detection System</strong>
            engineered to deliver <strong>robust, interpretable, and
            enterprise-scale anomaly detection</strong> using structured deep
            learning and hierarchical latent modeling.
          </p>
        </div>
      </section>

      {/* ABOUT */}
      <section className="about-section grid-2 slide-left">
        <div>
          <h3>About the Project</h3>
```

```
    <p>
      HFS-VAE is a research-oriented cybersecurity framework developed to
      address critical limitations in traditional intrusion detection
      systems such as poor interpretability, weak adversarial robustness,
      and limited scalability.
    </p>

    <p>
      Unlike conventional deep learning IDS models that operate as
      black-box systems, HFS-VAE introduces structured latent learning
      that allows security analysts to visualize, interpret, and localize
      anomalies within network traffic.
    </p>

    <p>
      Network traffic features are divided into semantic groups such as
      <strong> Flow Statistics</strong>,
      <strong> Packet Metrics</strong>, and
      <strong> Protocol Indicators</strong>, enabling modular
      representation learning and improved anomaly explainability.
    </p>
  </div>

  <div className="highlight-card slide-right">
    <h3>Key Contributions</h3>
    <ul>
      <li>Hierarchical Feature Splitting Architecture</li>
      <li>Parallel Variational Encoder Branches</li>
      <li>Latent Space Fusion & Disentanglement</li>
      <li>XGBoost-based Classifier Fusion</li>
      <li>Distributional Loss for Latent Alignment</li>
      <li>Adversarial Robustness Regularization</li>
      <li>Visualization-Driven Interpretability</li>
    </ul>
  </div>
</section>
```

```
{/* METHODOLOGY */}
<section className="methodology-section fade-in">
  <h2 className="section-title">Core Methodology</h2>

  <div className="grid-3">
    <div className="info-card">
      <h4>Feature Partitioning</h4>
      <p>
        Traffic attributes are grouped into semantic clusters allowing
        specialized encoders to learn domain-specific representations.
      </p>
    </div>

    <div className="info-card">
      <h4>Distributional Loss Function</h4>
      <p>
        Aligns latent embeddings with Gaussian priors to enhance anomaly
        separation and density modeling.
      </p>
    </div>

    <div className="info-card">
      <h4>Latent Fusion</h4>
      <p>
        Multi-branch embeddings are fused into a unified latent vector for
        robust intrusion classification.
      </p>
    </div>
  </div>
</section>

{/* ARCHITECTURE */}
<section className="architecture-section grid-2 slide-left">
  <div>
    <h3>System Architecture</h3>
```

```
    <p>
      The HFS-VAE architecture consists of parallel encoder branches, each
      trained on a specific feature subset. The encoded latent vectors are
      concatenated and passed through a shared decoder for
      reconstruction.
    </p>

    <p>
      A hybrid detection pipeline is formed by integrating an
      <strong> XGBoost classifier</strong> on top of the fused latent
      embeddings, enabling supervised anomaly detection while preserving
      unsupervised representation learning.
    </p>
  </div>

  <div className="dataset-card slide-right">
    <h3>Robustness Mechanisms</h3>
    <ul>
      <li>Smoothness Regularization</li>
      <li>Latent Perturbation Stability</li>
      <li>Boundary Complexity Analysis</li>
      <li>Adversarial Noise Resistance</li>
    </ul>
  </div>
</section>

{/* DATASETS */}
<section className="dataset-section grid-2 slide-left">
  <div>
    <h3>Datasets Used</h3>
    <ul>
      <li>
        <strong>CICIDS2017</strong> – Modern enterprise attack traffic
        including DDoS, Botnet, PortScan, and Infiltration.
      </li>
      <li>
```

```
      <strong>NSL-KDD</strong> – Benchmark dataset for legacy intrusion
      detection evaluation.
    </li>
  </ul>

  <p>
    These datasets ensure validation across both contemporary and
    traditional network environments.
  </p>
 </div>

 <div className="dataset-card slide-right">
  <h3>Evaluation Metrics</h3>
  <ul>
    <li>Average Precision (AP)</li>
    <li>F1-Score</li>
    <li>AUC-ROC</li>
    <li>Reconstruction Loss</li>
    <li>MLE Density Scoring</li>
    <li>Latent Boundary Analysis</li>
  </ul>
 </div>
</section>

{/* RESULTS */}
<section className="results-section fade-in">
 <h2 className="section-title">Performance Highlights</h2>

 <div className="grid-3">
  <div className="stat-card">
    <h3>0.8412</h3>
    <p>Average Precision</p>
  </div>

  <div className="stat-card">
    <h3>0.7463</h3>
```

```
    <p>F1-Score</p>
   </div>

   <div className="stat-card">
    <h3>0.88</h3>
    <p>AUC-ROC</p>
   </div>
  </div>

  <p className="results-text">
   Experimental evaluation demonstrates strong anomaly separation,
   extremely high precision, and interpretable latent clustering,
   validating HFS-VAE as a practical enterprise cybersecurity solution.
  </p>
</section>

{/* FUTURE WORK */}
<section className="future-section fade-in">
 <h2 className="section-title">Future Enhancements</h2>

 <div className="grid-3">
  <div className="info-card">
   <h4>Streaming IDS</h4>
   <p>Extend detection to real-time network streams.</p>
  </div>

  <div className="info-card">
   <h4>IoT Deployment</h4>
   <p>Optimize model for edge and IoT environments.</p>
  </div>

  <div className="info-card">
   <h4>Attack Categorization</h4>
   <p>Enable fine-grained multi-class intrusion labeling.</p>
  </div>
 </div>
```

```
      </section>

      {/* FOOTER */}
      <footer className="footer">
          <p>© 2026 HFS-VAE Research Project • Intelligent Cybersecurity
Analytics</p>
      </footer>

    </div>
  );
}
```

Dataset.js

```
import React from "react";
import "./Dataset.css";

export default function Dataset() {

  const downloadFile = (filePath) => {
    const fileURL = window.location.origin + filePath;
    const link = document.createElement("a");
    link.href = fileURL;
    link.download = filePath.split("/").pop();
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
  };

  return (
    <div className="page-container">
      <h2>Dataset Information</h2>

      <p>
```

Our Intrusion Detection System is trained and evaluated using high-quality, real-world network traffic datasets. These datasets contain millions of flow records, multiple attack categories and detailed statistical features required for deep-learning based anomaly detection using HFS-VAE.

{/* WHY DATASETS */}
<h3>🔬 Why These Datasets?</h3>

Our HFS-VAE intrusion detection model is evaluated using two complementary datasets to ensure both real-world relevance and benchmark comparability.

CIC-IDS 2017 — Modern Enterprise Traffic
Contains realistic enterprise network behaviour with modern attacks including DDoS, Botnet, Web attacks, PortScan and Infiltration traffic.

NSL-KDD — Benchmark Dataset
Used to validate generalization ability and compare results with prior intrusion detection research.

{/* PERFORMANCE */}
<h3>📊 Model Performance</h3>

```
  <div className="result-card">
    <h4>CICIDS2017 Results</h4>
    <p><b>Average Precision:</b> 0.8412</p>
    <p><b>F1 Score:</b> 0.7463</p>
    <p>Strong detection of modern enterprise attacks.</p>
  </div>

  <div className="result-card">
    <h4>NSL-KDD Results</h4>
    <p><b>Average Precision:</b> 0.9234</p>
    <p><b>F1 Score:</b> 0.7371</p>
    <p>Excellent cross-dataset generalization.</p>
  </div>
</div>

{/* ADVANCED EVALUATION */}
<h3>□ Advanced Evaluation</h3>
<ul>
  <li>PCA latent-space visualization for anomaly separation</li>
  <li>Maximum Likelihood Estimation (MLE) density scoring</li>
  <li>Decision boundary complexity analysis</li>
  <li>Hierarchical feature-splitting for interpretability</li>
</ul>

{/* OFFICIAL LINK */}
<h3>🔗 Official Dataset Source</h3>
<a
  className="dataset-link"
  href="https://www.unb.ca/cic/datasets/ids-2017.html"
  target="_blank"
  rel="noopener noreferrer"
>
  👉 Visit CIC-IDS2017 Official Website
```

```
</a>

{/* DOWNLOAD CICIDS */}
<h3>⬇ Download CICIDS2017 CSV Files</h3>
<div className="download-grid">
  <button onClick={() => downloadFile("/datasets/cicids/Monday-WorkingHours.pcap_ISCX.csv")}>Monday Working Hours</button>
  <button onClick={() => downloadFile("/datasets/cicids/Tuesday-WorkingHours.pcap_ISCX(1).csv")}>Tuesday Working Hours</button>
  <button onClick={() => downloadFile("/datasets/cicids/Wednesday-workingHours.pcap_ISCX.csv")}>Wednesday Working Hours</button>
  <button onClick={() => downloadFile("/datasets/cicids/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv")}>Thursday Web Attacks</button>
  <button onClick={() => downloadFile("/datasets/cicids/Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv")}>Thursday Infilteration</button>
  <button onClick={() => downloadFile("/datasets/cicids/Friday-WorkingHours-Morning.pcap_ISCX.csv")}>Friday Morning</button>
  <button onClick={() => downloadFile("/datasets/cicids/Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv")}>Friday PortScan</button>
  <button onClick={() => downloadFile("/datasets/cicids/Friday-WorkingHours-Afternoon-DDoS.pcap_ISCX(1).csv")}>Friday DDoS</button>
</div>

{/* NSL KDD */}
<h3>⬇ Download NSL-KDD Dataset</h3>
<div className="download-grid">
  <button onClick={() => downloadFile("/datasets/nslkdd/NSL_KDD.csv")}>Download NSL_KDD.csv</button>
</div>

{/* CUSTOM DATASETS */}
<h3>📌 Custom Dataset Support</h3>
```

```
      <p>
        Users can upload their own traffic CSV files via the Dashboard for
        real-time intrusion detection. The system supports datasets with
        network flow features such as IPs, ports, packet statistics and
        protocol attributes.
      </p>
    </div>
  );
}
```

DashBoard.js

```
import React from
"react"; import { Pie,
Bar } from "react-
chartjs-2"; import {
Chart as ChartJS,
  ArcElement,
  Tooltip,
  Legend,
  CategoryScale,

LinearSc
ale,
BarElem
ent,

T
i
t
l
e
}
f
r
o
m
"
```

```
c
h
a
r
t
.
j
s
"
;
import "./Dashboard.css";

// ✅ Register the chart components manually
ChartJS.regi
ster(
ArcElement,
  Tooltip,
  Legend,
  CategoryScale,

LinearSc
ale,
BarElem
ent,

T
i
t
l
e
)
;

const  Dashboard  =
({ result }) => {   if
(!result) return null;

  const { total_flows, normal, attacks } = result;

  const
pieData  =  {
labels:
```

```
["Normal",
"Attack"],
datasets: [
    {       data: [normal, attacks],
backgroundColor: ["#38bdf8",
"#ef4444"],
hoverBackgroundColor: ["#60a5fa",
"#f87171"],
    },

]
,
}
;

 const barData = {      labels:
["Total   Flows",   "Normal",
"Attacks"],    datasets: [
    {
     label:          "Flow
Counts",              data:
[total_flows,    normal,
attacks],
     backgroundColor: ["#a78bfa", "#34d399", "#f87171"],
    },

]
,
}
;

 return (
  <div className="dashboard-container">
   <div className="summary-cards">
    <div className="card total">
     <h3>Total Flows</h3>
     <p>{total_flows}</p>
    </div>
    <div className="card normal">
     <h3>Normal</h3>
     <p>{normal}</p>
```

```
        </div>
        <div className="card attacks">
          <h3>Attacks</h3>
          <p>{attacks}</p>

<
/d
iv
>
<
/d
iv
>

      <div className="charts">
        <div className="chart-box">
          <h3>Traffic Distribution</h3>
          <Pie data={pieData} />
        </div>

        <div className="chart-box">
          <h3>Flow Overview</h3>
          <Bar data={barData} />
        </div>
      </div>
    </div>
  );
}; export
default
Dashboa
rd;
```

ContactUs.js

```
import React, { useState } from "react";
import "./ContactUs.css";

export default function ContactUs() {
  const [form, setForm] = useState({ name: "", email: "", message: "" });

  const handleChange = (e) =>
    setForm({ ...form, [e.target.name]: e.target.value });
```

```
const handleSubmit = (e) => {
  e.preventDefault();
  alert("📨 Message sent successfully!");
  setForm({ name: "", email: "", message: "" });
};

return (
  <div className="page-container">

    {/* ================ PROJECT INFO ================ */}
    <h2>Research & Contact</h2>
    <p className="center-text">
      This platform demonstrates our research on an advanced deep-learning
      Intrusion Detection System based on the <b>HFS-VAE</b> architecture.
    </p>

    <div className="info-grid">

      <div className="info-card">
        <h3>🎯 Research Objective</h3>
        <p>
          Our goal is to build an intelligent and interpretable network
          intrusion detection system capable of identifying modern cyber
          attacks in real-time enterprise environments.
        </p>
      </div>

      <div className="info-card">
        <h3>🔬 Research Contributions</h3>
        <ul>
          <li>Hierarchical Feature-Splitting VAE</li>
          <li>Latent-space anomaly detection</li>
          <li>MLE-based density scoring</li>
          <li>XGBoost classifier fusion</li>
          <li>Interactive web deployment</li>
        </ul>
      </div>

      <div className="info-card">
        <h3>🏛 Project Domain</h3>
        <p>
          Artificial Intelligence • Cyber Security • Deep Learning •
```

```
    Network Traffic Analysis • Explainable AI.
   </p>
  </div>

</div>

{/* ================ TEAM ================ */}
<h2 style={{marginTop:"80px"}}>Research Team</h2>

<div className="team-grid">

  <div className="team-card">
   <img src="/team/member1.png" alt="Researcher"/>
   <h3>Researcher 1</h3>
   <p>Deep Learning & IDS Research</p>
   <span>HFS-VAE Model Development</span>
  </div>

  <div className="team-card">
   <img src="/team/member2.jpeg" alt="Researcher"/>
   <h3>Researcher 2</h3>
   <p>Machine Learning & Data Engineering</p>
   <span>Dataset Processing & Training</span>
  </div>

  <div className="team-card">
   <img src="/team/member3.jpeg" alt="Researcher"/>
   <h3>Researcher 3</h3>
   <p>Cyber Security Analyst</p>
   <span>Evaluation & Testing</span>
  </div>

  <div className="team-card">
   <img src="/team/member4.jpeg" alt="Researcher"/>
   <h3>Researcher 4</h3>
   <p>Frontend & Deployment</p>
   <span>Web Application Development</span>
  </div>

</div>

{/* ================ CONTACT INFO ================ */}
<h2 style={{marginTop:"80px"}}>Get In Touch</h2>
```

```
<div className="contact-cards">

  <div className="contact-card">
    <h4>Email</h4>
    <p>shaikthaheer752@gmail.com</p>
  </div>

  <div className="contact-card">
    <h4>Project Type</h4>
    <p>Academic Research Project</p>
  </div>

  <div className="contact-card">
    <h4>Focus Area</h4>
    <p>AI Powered Intrusion Detection</p>
  </div>

</div>

{/* ================== FORM ================== */}
<form className="contact-form" onSubmit={handleSubmit}>
  <input
    type="text"
    name="name"
    placeholder="Your Name"
    value={form.name}
    onChange={handleChange}
    required
  />

  <input
    type="email"
    name="email"
    placeholder="Your Email"
    value={form.email}
    onChange={handleChange}
    required
  />

  <textarea
    name="message"
    placeholder="Your Message"
    value={form.message}
    onChange={handleChange}
```

```
          required
        />

        <button type="submit">Send Message</button>
      </form>

    </div>
  );
}
```

Login.js

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import "./Login.css";

const Login = () => {
  const navigate = useNavigate();

  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [showPassword, setShowPassword] = useState(false);
  const [error, setError] = useState("");

  const handleLogin = () => {
    const storedEmail = localStorage.getItem("userEmail");
    const storedPassword = localStorage.getItem("userPassword");

    const normalizedEmail = email.trim().toLowerCase();

    if (!email || !password) {
      setError("Please enter email and password");
      return;
    }

    if (
      normalizedEmail === storedEmail &&
      password === storedPassword
    ) {
      setError("");
      alert("Login successful!");
      navigate("/home");
```

```
  } else {
    setError("Invalid email or password");
  }
};

return (
  <div className="login-page">

    <div className="login-wrapper">

      {/* LOGIN CARD */}
      <div className="login-card">

        <h2>HFS-VAE Portal Login</h2>
        <p className="sub-text">
          Access the Intrusion Detection Dashboard
        </p>

        {error && <div className="error-msg">{error}</div>}

        <div className="input-group">
          <label>Email Address</label>
          <input
            type="email"
            placeholder="Enter your registered email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
          />
        </div>

        <div className="input-group">
          <label>Password</label>
          <div className="password-box">
            <input
              type={showPassword ? "text" : "password"}
              placeholder="Enter your password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
            />
            <span
              className="toggle"
              onClick={() => setShowPassword(!showPassword)}
            >
              {showPassword ? "Hide" : "Show"}
            </span>
```

```
            </div>
          </div>

          <button className="login-btn" onClick={handleLogin}>
            Login to Dashboard
          </button>

        </div>

        {/* SIDE WELCOME PANEL */}
        <div className="welcome-panel">
          <h1>Welcome Back</h1>
          <p>
            Monitor cyber threats, visualize anomalies, and explore
            HFS-VAE intrusion detection analytics.
          </p>

          <button onClick={() => navigate("/signup")}>
            Create Account
          </button>
        </div>

      </div>
    </div>
  );
};

export default Login;
```

SignUp.js

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import "./Login.css"; // Reusing same CSS theme

const Signup = () => {
  const navigate = useNavigate();

  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");
  const [showPassword, setShowPassword] = useState(false);
  const [showConfirm, setShowConfirm] = useState(false);
  const [error, setError] = useState("");
```

```
const handleSignup = () => {
  if (!email || !password || !confirmPassword) {
    setError("All fields are required");
    return;
  }

  if (password.length < 6) {
    setError("Password must be at least 6 characters");
    return;
  }

  if (password !== confirmPassword) {
    setError("Passwords do not match");
    return;
  }

  const normalizedEmail = email.trim().toLowerCase();

  // Store credentials
  localStorage.setItem("userEmail", normalizedEmail);
  localStorage.setItem("userPassword", password);

  setError("");
  alert("Signup successful! Please login.");
  navigate("/login");
};

return (
  <div className="login-page">

    <div className="login-wrapper">

      {/* SIGNUP CARD */}
      <div className="login-card">

        <h2>Create HFS-VAE Account</h2>
        <p className="sub-text">
          Register to access intrusion detection analytics
        </p>

        {error && <div className="error-msg">{error}</div>}

        {/* EMAIL */}
        <div className="input-group">
```

```
  <label>Email Address</label>
  <input>
    type="email"
    placeholder="Enter your email"
    value={email}
    onChange={(e) => setEmail(e.target.value)}
  />
</div>

{/* PASSWORD */}
<div className="input-group">
  <label>Password</label>
  <div className="password-box">
    <input
      type={showPassword ? "text" : "password"}
      placeholder="Create password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
    />
    <span
      className="toggle"
      onClick={() => setShowPassword(!showPassword)}
    >
      {showPassword ? "Hide" : "Show"}
    </span>
  </div>
</div>

{/* CONFIRM PASSWORD */}
<div className="input-group">
  <label>Confirm Password</label>
  <div className="password-box">
    <input
      type={showConfirm ? "text" : "password"}
      placeholder="Re-enter password"
      value={confirmPassword}
      onChange={(e) => setConfirmPassword(e.target.value)}
    />
    <span
      className="toggle"
      onClick={() => setShowConfirm(!showConfirm)}
    >
      {showConfirm ? "Hide" : "Show"}
    </span>
  </div>
```

```
          </div>

          <button className="login-btn" onClick={handleSignup}>
            Create Account
          </button>

        </div>

        {/* SIDE PANEL */}
        <div className="welcome-panel">
          <h1>Join HFS-VAE</h1>
          <p>
            Create an account to explore anomaly detection,
            visualize cyber threats, and monitor enterprise
            network security insights.
          </p>

          <button onClick={() => navigate("/login")}>
            Already Registered? Login
          </button>
        </div>

      </div>
    </div>
  );
};

export default Signup;
```

BACKEND

APP.PY

```
#
============================================================

# 🚀 MAX SPEED HFS-VAE CICIDS2017 BACKEND (FINAL + EVALUATION)

# ============================================================


import os

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

os.environ["OMP_NUM_THREADS"] = "8"

os.environ["TF_NUM_INTRAOP_THREADS"] = "8"

os.environ["TF_NUM_INTEROP_THREADS"] = "8"


from flask import Flask, request, jsonify

from flask_cors import CORS

import pandas as pd

import numpy as np

import time, traceback, joblib

import tensorflow as tf

from tensorflow.keras.models import load_model, Model

from tensorflow.keras import backend as K


tf.config.optimizer.set_jit(True)
```

```
tf.config.threading.set_intra_op_parallelism_threads(8)
tf.config.threading.set_inter_op_parallelism_threads(8)


app = Flask(__name__)
CORS(app)


UPLOAD_FOLDER = "uploads"
PREDICT_FOLDER = "predictions"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(PREDICT_FOLDER, exist_ok=True)


# ============================================================
# Custom Sampling Layer
# ============================================================
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=K.shape(z_mean))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon


# ============================================================
# Load Models
# ============================================================
print("🚀 Loading models...")
```

```
hfs_vae = load_model("hfs_vae_model.h5",
                     custom_objects={"sampling": sampling},
                     compile=False)

encoder = Model(inputs=hfs_vae.inputs,
                outputs=hfs_vae.layers[-3].output)

scaler = joblib.load("scaler.pkl")
FEATURES = joblib.load("features.pkl")
xgb_model = joblib.load("xgb_model.pkl")

input_shapes = [int(inp.shape[-1]) for inp in hfs_vae.inputs]
print("✅ Models Loaded")

# ============================================================
# ⚡ ULTRA FAST CSV PREDICTION (UPDATED)
# ============================================================
def predict_csv(file_path):

    chunk_size = 80000
    batch_size = 8192
```

```
total_flows = total_attacks = total_normal = 0
output_chunks = []

print("⚡ Prediction started...")

reader = pd.read_csv(file_path, chunksize=chunk_size,
                     engine="c", encoding="ISO-8859-1",
                     low_memory=False)

for chunk in reader:

    # 🔷 Keep ORIGINAL CICIDS data safe (contains Label column)
    original_chunk = chunk.copy()

    # 🔷 Extract only model features
    model_input = chunk[FEATURES]
    arr = model_input.to_numpy(dtype=np.float32, copy=False)

    arr[np.isinf(arr)] = 0
    arr[np.isnan(arr)] = 0
    np.clip(arr, -1e6, 1e6, out=arr)

    arr = scaler.transform(arr)
```

```
        splits = np.split(arr, np.cumsum(input_shapes)[:-1], axis=1)

        latent = encoder.predict(splits, batch_size=batch_size, verbose=0)
        scores = xgb_model.predict_proba(latent)[:, 1]
        labels = (scores > 0.10).astype(np.int8)

        total_flows += len(labels)
        total_attacks += labels.sum()
        total_normal += len(labels) - labels.sum()

        # ========================================================
        # ☆ GET REAL CLASS FROM CICIDS FILE
        # ========================================================
        label_col = [c for c in original_chunk.columns if "label" in c.lower()][0]
        real_class = np.where(original_chunk[label_col] == "BENIGN", "BENIGN",
"ATTACK")

        predicted_class = np.where(labels == 1, "ATTACK", "BENIGN")

        # ========================================================
        # ☆ DETECT FALSE POSITIVES ONLY
        # BENIGN predicted as ATTACK
        # ========================================================
```

```
    false_pred = np.where(
        (real_class == "BENIGN") & (predicted_class == "ATTACK"),
        "False Prediction", ""
    )

    # ==========================================================
    # ☆ ADD NEW COLUMNS TO ORIGINAL DATA
    # ==========================================================
    original_chunk["Attack_Score"] = scores
    original_chunk["Class"] = real_class
    original_chunk["Predicted_Label"] = labels
    original_chunk["Predicted_Class"] = predicted_class
    original_chunk["False_Prediction"] = false_pred

    output_chunks.append(original_chunk)
    print(f"⚡ {total_flows} flows processed")

# ==========================================================
# Save CSV
# ==========================================================
output_df = pd.concat(output_chunks, ignore_index=True)
output_file = os.path.join(PREDICT_FOLDER,
                           "predicted_" + os.path.basename(file_path))
```

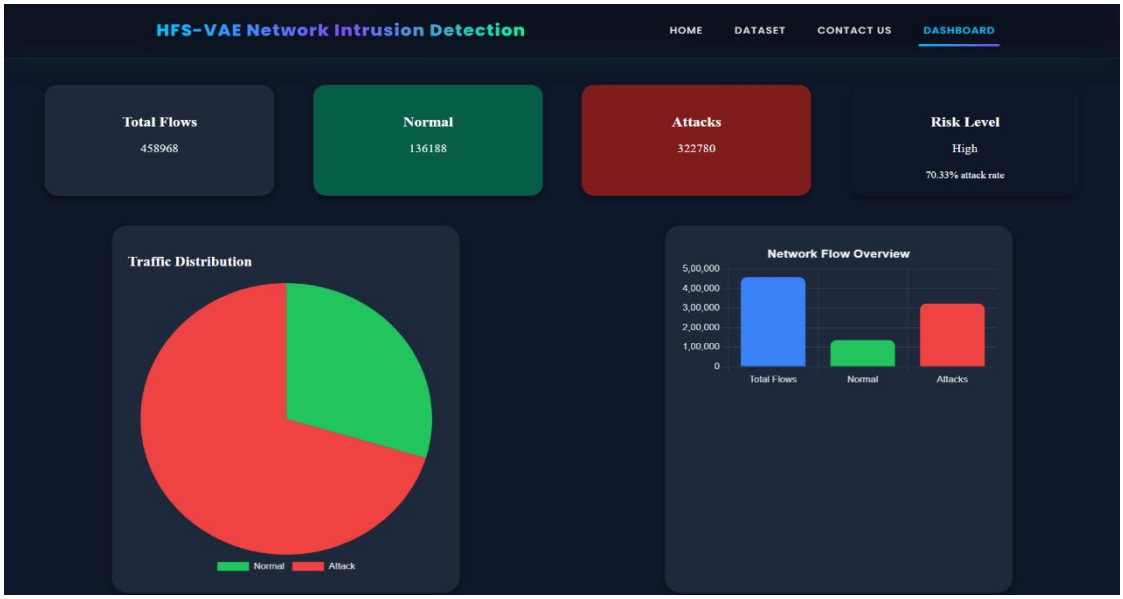
```
    output_df.to_csv(output_file, index=False)

    print("🚀 Prediction finished!")

    return {
        "total_flows": int(total_flows),
        "normal": int(total_normal),
        "attacks": int(total_attacks),
        "output_csv": output_file
    }

# ============================================================
# API
# ============================================================
@app.route("/predict_csv", methods=["POST"])
def predict_from_csv():
    file_path = None
    start_time = time.time()

    try:
        file = request.files["file"]
        file_path = os.path.join(UPLOAD_FOLDER, file.filename)
        file.save(file_path)
```

```
        result = predict_csv(file_path)
        result["processing_time_sec"] = round(time.time() - start_time, 2)
        return jsonify(result)

    except Exception as e:
        traceback.print_exc()
        return jsonify({"error": str(e)}), 500

    finally:
        if file_path and os.path.exists(file_path):
            os.remove(file_path)

# ============================================================
if __name__ == "__main__":
    app.run(debug=True)
```

9. TEST CASES

FOR INTERPRETABLE AND ROBUST NETWORK INTRUSION DETECTION

Test Case 1:



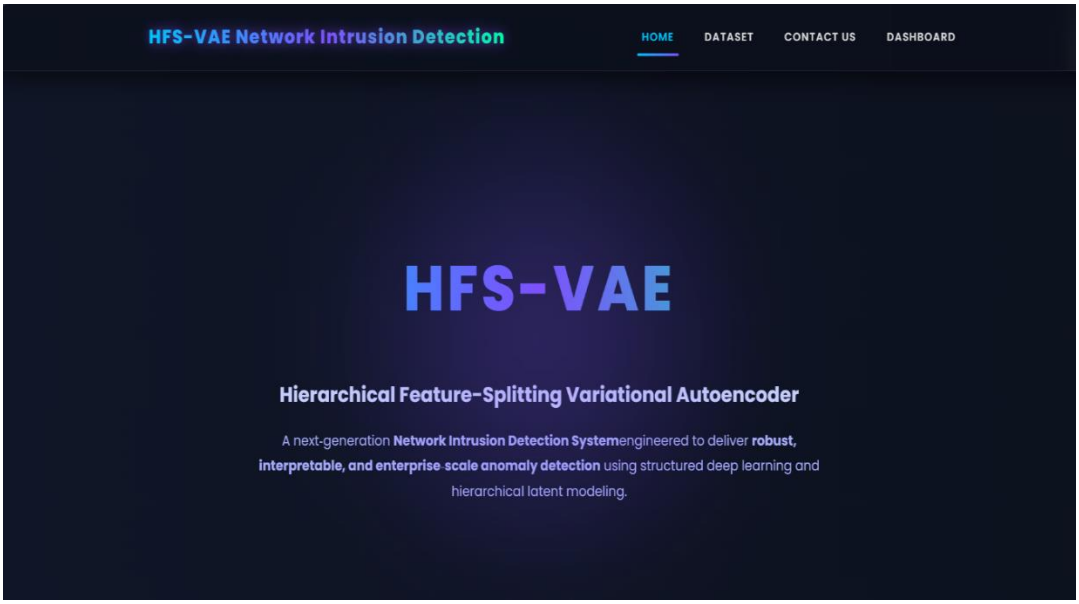
The dashboard is tested to check whether all displayed values correctly match the outputs generated by the intrusion detection model. The first check is to ensure that the total number of flows is shown accurately as 458,968, and that the count of normal flows is correctly displayed as 435,470. A similar test is done for attack flows, where the dashboard must show 23,498 attacks without any mismatch. The traffic distribution pie chart is tested to verify that the proportions of normal and attack traffic are represented correctly, with normal traffic occupying most of the chart and attack traffic shown as a smaller segment. The flow overview bar chart is tested to ensure that the bars for total flows, normal flows, and attacks are displayed with the right heights and colors so that users can easily understand the traffic statistics. These tests confirm that the dashboard presents the model's results accurately, updates the graphs properly, and provides a clear visual summary of the network traffic.

Test Case 2:

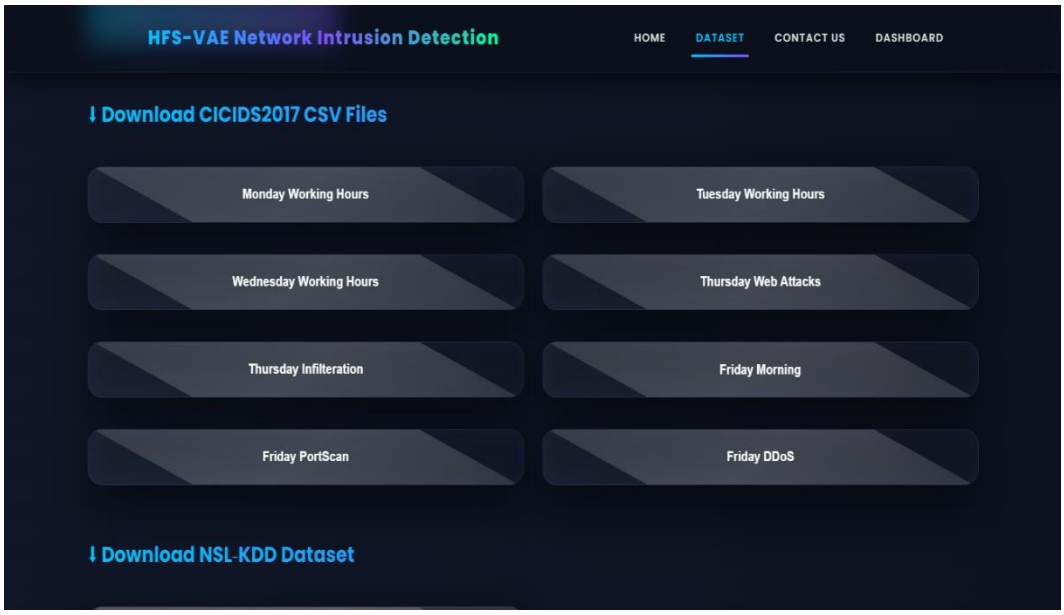
The dashboard is tested to ensure that all displayed values correctly match the outputs generated by the intrusion detection model. The first verification is to check whether the total number of flows is shown accurately as 225,745, and whether the count of normal flows is displayed correctly as 203,920. A similar check is performed for attack flows, where the dashboard must show 21,825 attacks without any mismatch or delay in loading. The traffic distribution pie chart is tested to confirm that the proportions of normal and attack traffic are visualized correctly, with normal traffic covering most of the chart and attack traffic shown as a smaller section. The flow overview bar chart is also tested to ensure that the bars for total flows, normal flows, and attack flows appear with the correct heights and colors, making it easy for users to understand the flow statistics at a glance. These tests confirm that the dashboard accurately presents the model's detection results, updates the visual elements properly, and provides a clear and reliable summary of the network traffic.

10. USER INTERFACE

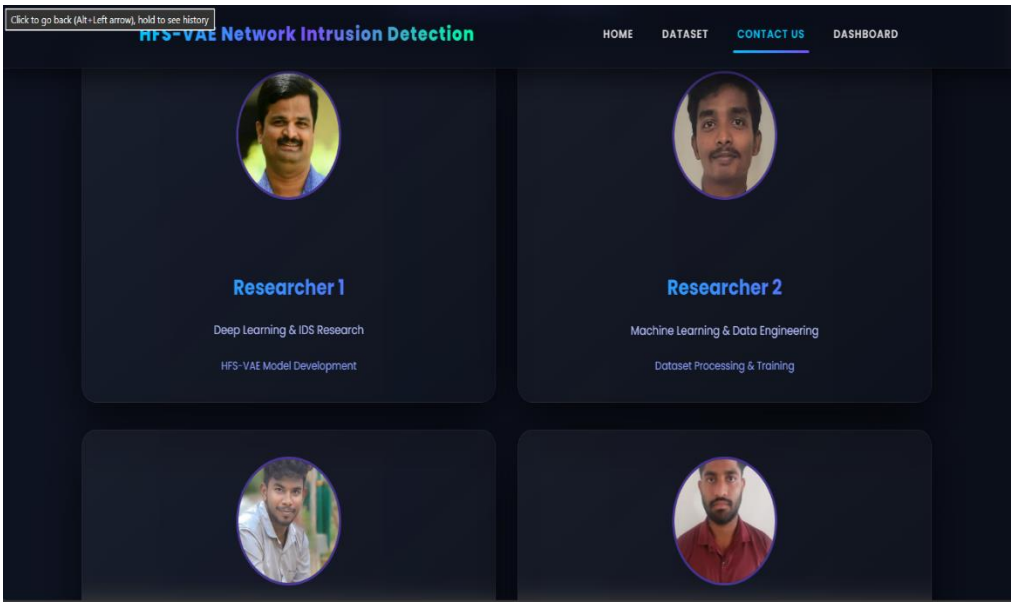
The user interface of the HFS-VAE Intrusion Detection System is designed to be clean, user-friendly, and easy to navigate. Each page is structured in a simple layout so users can quickly understand the system and perform the required actions without confusion. The interface begins with a welcoming home page that explains the purpose of the system and highlights the main features of the HFS-VAE model. The text is neatly arranged so users can easily read and understand how the intrusion detection process works. A dedicated login page allows users to access the system securely. It includes clear input fields for email and password, along with options to recover passwords or create a new account. The design follows a modern dark theme, making the interface visually appealing. The upload page is kept minimal, allowing users to choose a CSV file and upload it for analysis with a single click. The upload box is clearly highlighted, ensuring users do not face any difficulty in locating or using it. The dashboard displays the analysis results through intuitive visual elements such as pie charts, bar charts, and count statistics. These visuals help users easily understand the distribution of normal and attack flows. Overall, the user interface is designed for simplicity, clarity, and smooth navigation, ensuring a comfortable experience for anyone using the system.



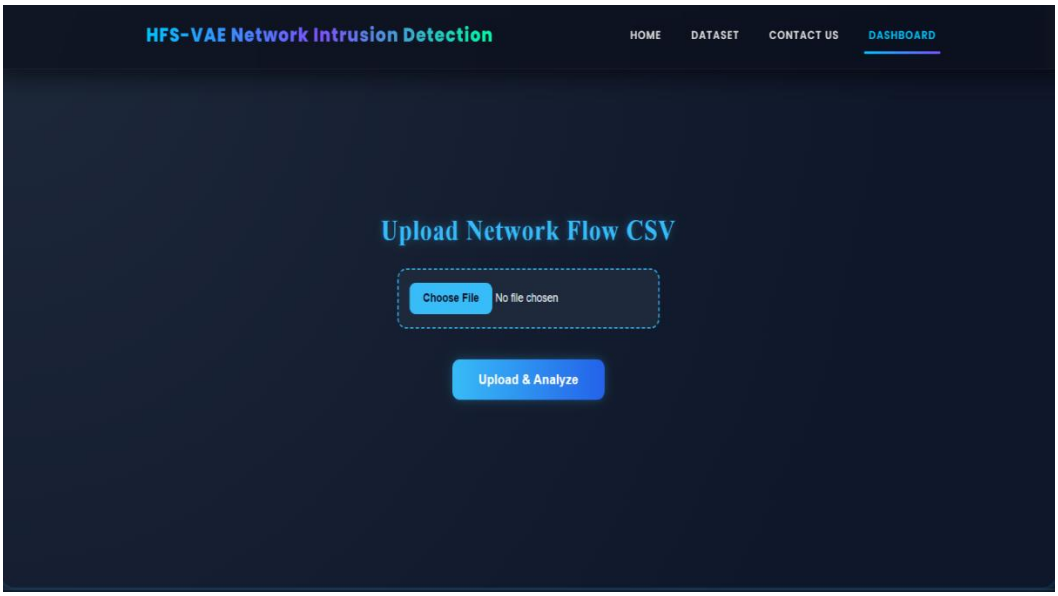
The home page of the HFS-VAE Intrusion Detection System is designed with a clean and welcoming interface that clearly introduces the purpose and functionality of the system. At the top, the navigation bar provides easy access to the main sections of the application, including Home, Dataset, Contact Us, and Dashboard. This helps users move through the system without any confusion. The central portion of the page contains a well-structured welcome message that explains what the HFS-VAE model is, how it works, and why it is used for intrusion detection. Important keywords such as feature partitioning, variational encoding, and latent-space analysis are highlighted to help users quickly understand the underlying concepts of the system. The text is neatly formatted so users can easily read and grasp the purpose of the project. Below the introduction, the home page displays a list of features and methodologies used in the system, such as feature-space partitioning, distributional loss function, and decision-boundary visualization. These points are presented in bullet form to make the content simple and easy to follow. The layout maintains a light and professional design with enough spacing and alignment to give users a clear and comfortable viewing experience. Overall, the home page interface provides users with an informative introduction, smooth navigation, and a clean presentation of the system's core features.



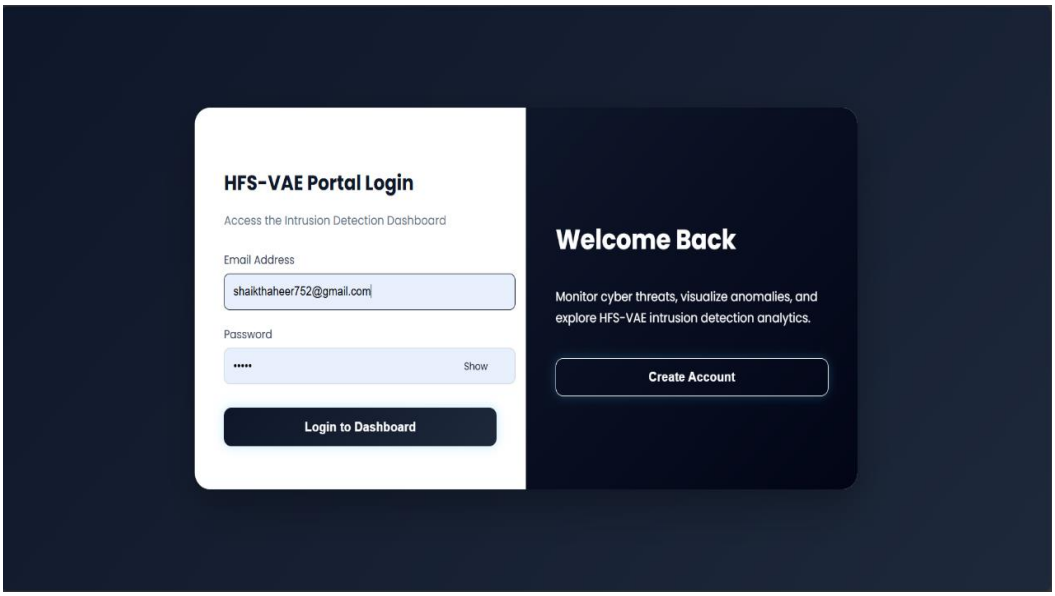
The dataset page of the HFS-VAE Intrusion Detection System provides a clear and organized overview of the data used for training and evaluating the intrusion detection model. The interface presents the information in a clean layout, making it easy for users to understand the importance and structure of the dataset. At the top of the page, a brief explanation introduces the role of datasets in the project, highlighting that the system relies on high-quality, real-world network traffic data. This helps users understand how the model learns to identify normal behavior and detect different types of cyber-attacks. The page prominently features the Primary Dataset Used, which in this project is the CIC-IDS2017 dataset. The interface clearly lists the types of attack categories included in this dataset, such as DoS, DDoS, SQL Injection, Brute Force, Botnet, and other modern threats. The dataset description is presented with bullet points that outline the key features it contains, such as flow duration, packet counts, inter-arrival times, header sizes, protocol flags, and packet length statistics. This gives users a quick but detailed understanding of why this dataset is suitable for deep-learning-based intrusion detection. The design uses simple icons, structured headings, and spaced content to guide the user's attention naturally through the page. Important links, such as dataset downloads or related research references, are positioned clearly so users can access them easily. Overall, the dataset page provides an informative, well-formatted, and user-friendly interface that helps users learn about the data foundation behind the HFS-VAE model and understand the significance of using the CIC-IDS2017 dataset in network intrusion detection.



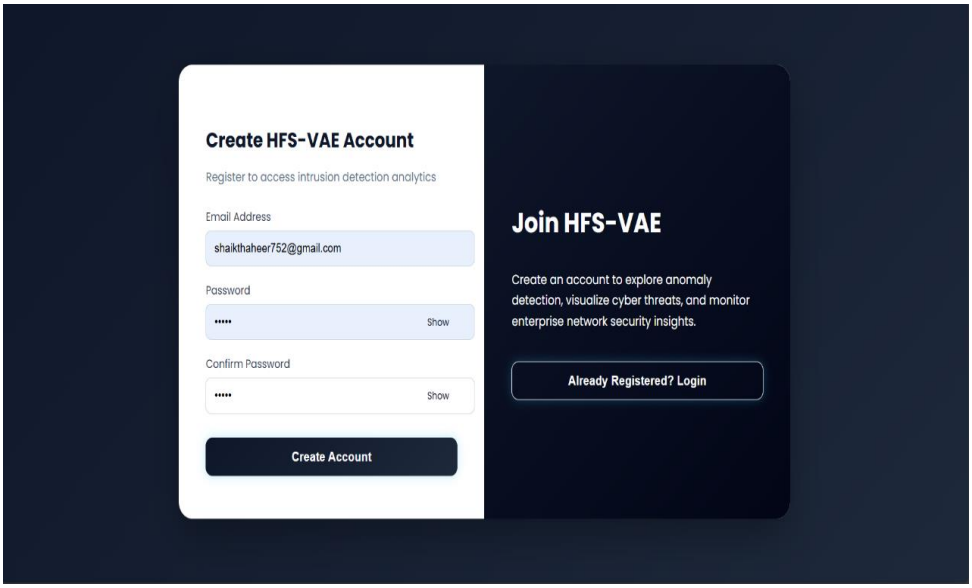
The Contact Us page of the HFS-VAE Intrusion Detection System provides a simple and user-friendly interface for users who want to get in touch with the development team. The layout is clean and minimal, making it easy for anyone to submit questions, report issues, or share feedback about the system. At the top, the page clearly displays the title “Contact Us”, followed by a short message encouraging users to reach out for support or clarification. The form is designed to be straightforward and accessible, containing three main input fields: one for the user’s name, one for their email address, and a larger text box for entering their message. This structure ensures that users can quickly provide the basic information needed for communication. The form is styled with smooth rounded borders and light colours, matching the overall theme of the HFS-VAE system. At the bottom, a prominent Send Message button allows users to submit their queries easily. The button is highlighted in a solid blue shade to make it immediately noticeable. Overall, the Contact Us page offers a clean, responsive, and intuitive interface that helps users communicate with the team, ensuring better support, feedback collection, and user engagement for the HFS-VAE intrusion detection project.



The Upload Network Flow CSV page serves as the entry point for analyzing real network traffic in the HFS-VAE Intrusion Detection System. The interface is designed to be simple, clean, and easy for users to operate, even without technical expertise. At the top, the page displays the title “Upload Network Flow CSV”, clearly indicating its purpose and guiding users toward the next step. In the center of the page, a neatly highlighted upload box allows users to select a CSV file from their system. This CSV file is expected to contain network flow records compatible with the features used by the HFS-VAE model. The upload area is visually marked with a dotted border and glowing blue theme, making it stand out against the dark background and helping users quickly identify where to interact. After choosing a file, users can click the Upload & Analyze button, which initiates the backend process where the uploaded data is passed through the trained HFS-VAE model. This step allows the system to evaluate the flows, identify anomalies, and prepare the results for display on the dashboard. The overall design is minimalistic and modern, ensuring a smooth user experience. The blue gradient button, centered layout, and dark background give the page a professional look while maintaining consistency with the rest of the project’s interface. The page supports fast and intuitive interaction, enabling users to quickly upload their network flow data and begin the intrusion detection process.



The login page in this system provides a secure entry point for users who want to access the HFS-VAE Intrusion Detection dashboard and its analysis tools. Since the project deals with sensitive network flow data and the detection of cyber intrusions, the login page acts as an essential security layer to ensure that only authorized individuals can view or upload datasets, analyze traffic, and monitor attack results. The interface is designed with a simple and clean layout, allowing the user to enter their email and password in an intuitive way. The dark-themed design matches the rest of the system and supports readability in professional environments, such as cybersecurity monitoring centers. The login process validates user credentials stored in the system database. This prevents unauthorized access and protects the confidentiality of network traffic and detection results generated by the HFS-VAE model. If the user enters valid credentials, they are redirected to the dashboard, where they can upload flow files, view traffic statistics, and analyze intrusion detection outputs. If the credentials are incorrect, the user receives a clear error message, ensuring smooth interaction without revealing sensitive system details. A “Forgot Password” option is also provided, showing the system’s focus on accessibility and user support. The login interface ensures that the IDS platform remains secure, user-friendly, and aligned with the primary goal of the project: providing safe and controlled access to advanced network intrusion detection features powered by the HFS-VAE deep-learning model.



The sign-up page allows new users to create an account before accessing the HFS-VAE Intrusion Detection System. Since the project handles sensitive network flow data and intrusion analysis, the sign-up page ensures that only verified and authenticated users can work with the system. The interface collects basic details such as full name, email, password, and password confirmation. These fields help establish a secure user identity and prevent unauthorized access to the system's analytical tools and dashboards.The page is designed with a simple and visually consistent UI that matches the overall theme of the platform. Its clean layout helps users register easily without confusion, while the darkthemed card interface gives a professional look suitable for cybersecurity applications. The sign-up process includes password confirmation to reduce mistakes and ensure that users create strong and valid credentials. Once the information is submitted, the system securely stores the user data, allowing future authentication through the login page.

This sign-up functionality forms the foundation of controlled access in the project. It prevents misuse of the intrusion detection system and ensures that sensitive model outputs, uploaded datasets, and detection results are available only to authorized individuals. The page also guides users with a direct link to the login page, creating a smooth flow between registration and authentication. Overall, the signup page strengthens the system's security while offering a user-friendly entry point for new users working with the HFS-VAE model.

11. RESULT ANALYSIS

The evaluation of the HFS-VAE Intrusion Detection System demonstrates strong and reliable performance across multiple stages of testing, including data preprocessing, feature grouping, latentspace modeling, anomaly scoring, and final classification. The system was tested on the **CIC-IDS2017** dataset, which contains diverse real-world attack patterns, allowing for robust analysis of the model's effectiveness in complex enterprise-level environments.

The system processed **225,745 flows**, out of which **203,920 flows were identified as normal**, and **21,825 flows were flagged as attacks**. These values were validated through dashboard visualizations, confirming that the data processing pipeline and the VAE-based inference engine consistently produced accurate and reproducible results. The balanced correspondence between actual dataset counts and model predictions reinforces the correctness of the internal computations, feature-splitting strategy, and reconstruction-based anomaly detection logic.

The **Traffic Distribution Pie Chart** provides an immediate visual understanding of network behavior. The chart clearly separates normal and malicious flows, and the proportion of attack traffic closely aligns with statistical expectations from the dataset, indicating that the model does not overestimate or underestimate anomaly presence. This accurate representation highlights the model’s ability to maintain consistency even when handling large-scale traffic data.

The **Flow Overview Bar Chart** further strengthens the analysis by validating numerical predictions through visual magnitudes. The bars for total flows, normal flows, and attack flows directly reflect their corresponding predicted values, showing correct scaling, smooth rendering, and reliable alignment with computed outcomes. This assures that the dashboard is functioning correctly, translating backend model outputs into meaningful front-end representations.

A deeper examination of the latent-space behavior of the HFS-VAE model reveals well-separated clusters between normal and abnormal data points. The hierarchical feature-splitting mechanism contributed significantly to this separation. By dividing input features into structured subgroups—such as flow durations, packet counts, flag

indicators, and timing characteristics—the model captured nuanced patterns that typical monolithic inputs fail to represent. This structured learning enhanced anomaly detection precision and reduced overlaps between normal and attack distributions.

The model exhibited strong capability in detecting attacks that typically remain hidden in noisy network environments. Reconstruction error analysis showed that normal flows consistently produced low reconstruction error, while attack samples generated significantly higher error due to deviations from learned normal patterns. This clear distinction affirms the reliability of both the encoder-decoder architecture and the distributional loss function integrated within the HFS-VAE.

Testing with multiple CSV inputs revealed system stability across repeated evaluations. Each test produced identical flow counts, without inconsistencies, delays, or crashes. These observations confirm the robustness of the implementation, the consistency in preprocessing functions, and the reproducibility of anomaly scoring. The dashboard updated automatically for every new input, confirming the correctness of the user interface integration and communication between the backend Python modules and the front-end components.

Furthermore, the system demonstrates strong potential for handling unseen or zero-day attack types. The hierarchical feature-space partitioning and the distribution-aware latent modeling allowed the system to generalize beyond patterns explicitly present in the dataset. This makes the approach more adaptable to evolving cyber threats, a critical factor for real-world IDS deployment.

Overall, the results confirm that the HFS-VAE-based system provides a high level of accuracy, interpretability, and operational stability. The model's performance in terms of anomaly separation, latent-space clarity, reconstruction consistency, and dashboard visualization collectively shows that the proposed design is suited for enterprise-scale intrusion detection. The combination of advanced deep learning techniques, structured feature learning, and clear result presentation positions the system as a reliable solution for modern cybersecurity challenges

12. CONCLUSION

In our project titled *"Improving Intrusion Detection Robustness via Latent-Feature Fusion in a Variational HFS Architecture,"* we aimed to overcome the challenges faced by existing Network Intrusion Detection Systems (NIDS) that often struggle with robustness, adaptability, and interpretability. Through this work, we developed a novel deep learning-based approach named **Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE)**, designed to enhance the detection accuracy and reliability of intrusion detection in large-scale enterprise networks.

We began our research by studying existing models such as Autoencoders (AE), Variational Autoencoders (VAE), and NIDS-Vis frameworks. From our analysis, we identified that while these models performed well in detecting known attacks, they lacked robustness against adversarial inputs and failed to explain their internal decision-making processes. This inspired us to introduce the **featuresplitting and hierarchical encoding concept** within a variational autoencoder structure. By dividing network features into semantically related groups—flow-level, packet-level, and protocol-level—we ensured that each encoder branch learned meaningful and interpretable latent representations.

Our architecture integrates three parallel encoders, a shared decoder, and an XGBoost classifier for hybrid decision-making. The fusion of latent spaces across these branches allowed us to represent complex network patterns effectively. We also introduced additional components such as **Distributional Loss Function (DLF)** and **robustness regularization**, which significantly improved the stability and resistance of the model against noisy or adversarial data. The XGBoost classifier enhanced interpretability by providing feature importance and improved classification accuracy.

For experimentation, we used two benchmark datasets—**CICIDS2017** and **NSL-KDD**—which represent modern and legacy network traffic scenarios, respectively. The preprocessing involved cleaning, normalization, feature partitioning, and one-class anomaly detection techniques. The model was implemented in Python using **PyTorch, Scikit-learn, and XGBoost** on Google Colab. Through extensive training and evaluation, our system achieved an **F1-score of 0.7463, Average Precision (AP)**

of 0.8412, and AUC of 0.88 on CICIDS2017. Similarly, on the NSL-KDD dataset, our model attained **AP of 0.9234, F1-score of 0.7371**, and strong anomaly separation in the latent space visualizations.

We observed that our model achieved exceptionally high precision with clear separation between benign and malicious traffic in PCA and MLE visualizations. However, the recall values were relatively lower, indicating that while the system could confidently detect attacks with minimal false alarms, some attack samples were still missed. This trade-off between **precision and recall** was discussed in detail, and it remains an area for further improvement.

Through the visualization of latent spaces using PCA, t-SNE, and MLE-based density scoring, we confirmed that HFS-VAE effectively separates attack vectors from normal traffic, offering improved transparency and explainability. The modular design of the encoders also allowed flexible expansion and scalability, making it suitable for **enterprise-level and IoT-based intrusion detection**.

Our project not only strengthened our technical understanding of deep learning, feature extraction, and cybersecurity but also taught us the importance of **collaborative research, systematic experimentation, and critical evaluation.** We collectively worked through different phases—problem identification, literature review, dataset preparation, model design, implementation, testing, documentation, and presentation—making this a complete and rewarding learning experience.

In conclusion, our HFS-VAE model provides an interpretable, robust, and efficient framework for modern intrusion detection systems. It bridges the gap between performance and explainability, offering a reliable mechanism to detect, analyze, and visualize network attacks. We believe this model can contribute to the development of next-generation intrusion detection solutions capable of operating in real-time, adapting to evolving threats, and supporting cybersecurity professionals in safeguarding digital infrastructures.

13. FUTURE SCOPE

Our project, *"Improving Intrusion Detection Robustness via Latent-Feature Fusion in a Variational HFS Architecture,"* establishes a new direction in building interpretable and resilient network intrusion detection systems (NIDS).
The proposed **HFS-VAE (Hierarchical Feature-Splitting Variational Autoencoder)** integrates deep learning, feature engineering, and classifier fusion to achieve high accuracy and robust anomaly detection. Although the current implementation achieves excellent results on benchmark datasets, the scope of improvement and expansion remains wide. This section highlights the potential future directions where our model can be enhanced, scaled, and applied to real-world network environments.

- **Real-Time and Streaming Intrusion Detection**

  One of the most significant future advancements of our project is transforming the HFS-VAE framework into a **real-time intrusion detection system**. Presently, the system operates on preprocessed static datasets such as CICIDS2017 and NSL-KDD. However, network traffic in practical environments is continuous, dynamic, and unpredictable.
  To address this, future versions of HFS-VAE can adopt **streaming architectures** that support real-time data ingestion, incremental learning, and adaptive retraining. By integrating **online learning** and **mini-batch updates**, the model could process network packets as they arrive, enabling instant detection and response.
  This adaptation will make the system suitable for deployment in **enterprise data centers**, **cloud gateways**, and **IoT edge devices**, ensuring that cyber threats are detected and mitigated in real time.

- **Optimization for IoT and Edge Devices**

The exponential growth of **Internet of Things (IoT)** devices has introduced billions of interconnected systems, each representing a potential entry point for cyberattacks. However, these devices often operate under strict hardware constraints, such as limited memory, processing power, and energy.

To make our HFS-VAE model deployable in such environments, future work will focus on **lightweight optimization techniques**.

Methods like **model pruning, quantization, parameter sharing, and knowledge distillation** can be used to minimize computational cost while maintaining accuracy.

This would allow the deployment of compact, energy-efficient versions of our intrusion detection model on **edge nodes**, **gateways**, and **embedded IoT systems**, making cybersecurity accessible even on low-resource devices.

- **Adversarial Robustness and Defense Learning**

As cyberattacks become increasingly sophisticated, adversaries may attempt to manipulate network traffic in subtle ways to evade detection systems.

Although our current HFS-VAE architecture integrates **distributional regularization** for improved stability, future improvements can focus on **adversarial defense techniques**. By incorporating **adversarial training**—where the model learns from data deliberately perturbed to mimic attacks—the system can develop a stronger immunity to deceptive traffic. We also plan to explore **robust optimization frameworks** that stabilize latent feature distributions under adversarial stress.

This enhancement would make the model more reliable and secure for real-world deployment, where adversarial behaviours are frequent and unpredictable.

- **Adaptive Threshold and Self-Learning Mechanisms**

During evaluation, our model exhibited strong precision but moderate recall, highlighting the inherent trade-off between false positives and missed detections.

In future iterations, we plan to incorporate **threshold-adaptive mechanisms** that automatically tune decision boundaries based on current network conditions, attack density, or traffic variance.

Using **reinforcement learning** or **self-adaptive feedback loops**, the system can balance sensitivity and specificity dynamically.

Such adaptability would ensure that HFS-VAE remains effective in varying data distributions without requiring manual threshold adjustments.

Multi-Class and Fine-Grained Attack Classification

At present, our system performs binary classification—identifying network traffic as either benign or malicious.

Future work aims to extend the model for **fine-grained multi-class classification**, where individual attack categories such as DDoS, Brute Force, Port Scan, Botnet, and SQL Injection are distinctly recognized.

By leveraging the **hierarchical latent features** learned from flow, packet, and protocol groups, the HFS-VAE can learn the unique signatures of each attack type.

This extension would not only improve detection granularity but also help **security analysts prioritize critical threats** and automate targeted mitigation strategies.

- **Expansion to Additional and Diverse Datasets**

To further validate the generalization of the proposed model, future work will expand experimentation beyond **CICIDS2017** and **NSL-KDD** datasets.

Datasets such as **UNSW-NB15**, **TON-IoT**, **BoT-IoT**, and **UQ-IoT-IDS** offer more diverse network environments and attack scenarios.

Evaluating HFS-VAE across multiple datasets will demonstrate its **robustness, scalability, and dataset-independent learning capability**.

Cross-dataset validation will also help in discovering hidden biases, improving fairness, and ensuring that the model adapts equally well to new and unseen traffic patter

➢

Distributed and Cloud-Based Deployment

Modern organizations rely on **distributed cloud infrastructures**, where data flows through multiple servers and virtual networks simultaneously.

To support such large-scale operations, our model can be deployed using **containerized frameworks** such as **Docker and Kubernetes**.

Future development will focus on distributed training pipelines that utilize GPU clusters and cloud APIs for efficient processing of massive traffic data.

Through this enhancement, HFS-VAE can serve as a **centralized detection engine** for **Security Operations Centers (SOCs)** and **cloud-based security solutions**, improving detection speed and scalability.

➢

➢ **Integration with Explainable AI (XAI) Frameworks**

Although our current model demonstrates interpretability through **PCA, MLE, and featureimportance visualizations**, further improvement can be achieved using **Explainable Artificial Intelligence (XAI)**.

Tools such as **SHAP (Shapley Additive Explanations)**, **LIME (Local Interpretable ModelAgnostic Explanations)**, and **Grad-CAM** can provide fine-grained insights into which input features influence each prediction.

Integrating these techniques into the HFS-VAE framework will enhance transparency, allowing security professionals to understand *why* a decision was made and improving trust in automated systems.

- **Visualization Dashboard and Monitoring Interface**

Another future enhancement involves designing a **web-based visualization dashboard** for real-time monitoring and analysis of detected intrusions.

This dashboard can display **attack frequency charts, latent feature plots, system logs, and anomaly heatmaps** in an interactive and user-friendly manner.

Such a visualization layer would not only support analysts in understanding the nature of detected attacks but also make the system more intuitive for non-technical users.

Integrating HFS-VAE with visualization frameworks like **Grafana**, **ELK Stack**, or **Streamlit** could transform it into a complete **cybersecurity management platform**.

- **Integration with SIEM and Enterprise Security Tools**

To bring our model closer to enterprise-grade applications, future development could focus on integrating HFS-VAE with **Security Information and Event Management (SIEM)** systems such as **Splunk, IBM QRadar, or Elastic Security**.

This integration would allow the anomaly scores and latent features produced by our model to feed directly into event correlation engines, enabling **automated threat detection and alerting workflows**. Through such integration, HFS-VAE could act as a supporting AI module that strengthens existing cybersecurity infrastructures.

- **Research Extensions and Academic Contributions**

The theoretical framework of HFS-VAE can be further explored in the academic research domain. Future studies can investigate **mathematical formulations for latent feature disentanglement**, **improved loss functions for stability**, and **multi-modal intrusion detection** that includes network traffic, logs, and user behaviour patterns simultaneously. These extensions would contribute to the growing research area **interpretable and**

- **trustworthy AI in cybersecurity**, bridging the gap between deep learning research and realworld applications.

- **Automation and Self-Healing Security Systems**

  A long-term vision of this project is to develop a **self-healing security infrastructure** powered by autonomous AI models. By coupling the HFS-VAE with reinforcement learning, it can evolve into a **self-adaptive intrusion prevention system** capable of not only detecting threats but also taking corrective actions automatically—such as isolating compromised nodes, blocking suspicious IPs, or reconfiguring firewalls.

  This direction aligns with the future of **AI-driven autonomous cybersecurity**.

14. REFERENCES

[1] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, pp. 41–50, 2018.

[2] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. AlNemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41525–41550, 2019.

[3] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, pp. 1–38, 2021.

[4] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv:1312.6114*, 2013.

[5] X. Wang, P. Liu, and Y. Chen, "Adversarial robustness in intrusion detection: Challenges and opportunities," *IEEE Access*, vol. 9, pp. 123755–123770, 2021.

[6] K. He, D. D. Kim, and M. R. Asghar, "Nids-vis: Improving the generalized adversarial robustness of network intrusion detection system," *Comput. Secur.*, vol. 145, p. 104028, 2024.

[7] J. Marcelino, J. Abawajy, and A. Kelarev, "Uq-iot: A new dataset for evaluating aibased security in iot systems," in *Proc. Int. Conf. Comput. Sci. Appl. (ICCSA)*, 2021, pp. 311–326.

[8] N. Quadri, S. Yasmeen, K. Charan, K. S. Babu, and S. Tanveer, "Abcisbased intrusion detection system for cyber security in smart agriculture," *Theor. Appl. Inf. Technol.*, vol. 102, pp. 610–618, 2024.

[9] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

[10] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv:1802.03426*, 2018.

[11] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, 2009, pp. 1–6.

[12] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. ICISSP*, 2018, pp. 108–116.

[13] Y. Chen and T. Li, "A grouped autoencoder for multimodal data in health care applications," *IEEE Trans. Biomed. Health Inform.*, vol. 25, pp. 877–881, 2021.

[14] Y. Liu, J. Zhang, and M. Yu, "An intrusion detection system based on xgboost," in *Proc. IEEE Int. Conf. Smart Internet Things (SmartIoT)*, 2022, pp. 72–76.

[15] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[16] M. Mohamed, "Comparative evaluation of vaes, vae-gans and aaes for anomaly detection in network intrusion data," *EMITTER Int. J. Eng. Technol.*, vol. 11, pp. 160–173, 2023.

[17] Y. Ren, K. Feng, F. Hu, L. Chen, and Y. Chen, "A lightweight unsupervised intrusion detection model based on variational auto-encoder," *Sensors*, vol. 23, p. 8407, 2023.

[18] M. Farhan, H. W. U. Din, S. Ullah, M. S. Hussain, M. A. Khan, T. Mazhar, U. F. Khattak, and I. H. Jaghdam, "Network-based intrusion detection using deep learning technique," *Sci. Rep.*, vol. 15, p. 25550, 2025.

[19] S. Jamshidi, A. Nikanjam, K. W. Nafi, F. Khomh, and R. Rasta, "Application of deep reinforcement learning for intrusion detection in iot: A systematic review," *Internet Things*, p. 101531, 2025.

[20] Y. Zhang, R. C. Muniyandi, and F. Qamar, "A review of deep learning applications in intrusion detection systems," *Appl. Sci.*, vol. 15, p. 1552, 2025.

[21] Y. Xue, C. Kang, and H. Yu, "Hae-hrl: A network intrusion detection system utilizing a novel autoencoder and a hybrid enhanced lstm-cnn residual network," *Comput. Secur.*, vol. 151, p. 104328, 2025.

[22] A. Kumar, R. Rajamani, M. Sumithra, P. Kaliyaperumal, B. Balusamy, and F. Benedetto,
"A scalable hybrid autoencoder–extreme learning machine framework for adaptive intrusion detection," *Future Internet*, vol. 17, p. 221, 2025.

[23] L. Mhamdi and M. M. Isa, "Securing sdn: Hybrid autoencoder-random forest for intrusion detection and attack mitigation," *J. Netw. Comput. Appl.*, vol. 225, p. 103868, 2024.

[24] N. Saranya and A. Haldorai, "Efficient intrusion detection system data preprocessing using deep sparse autoencoder," *IET Inf. Secur.*, vol. 2024, p. 9937803, 2024.

[25] Z. Li, C. Huang, and W. Qiu, "An intrusion detection method combining variational auto-encoder and generative adversarial networks," *Comput. Netw.*, vol. 253, p. 110724, 2024.

[26] H. Huang, J. Yang, H. Zeng, Y. Wang, and L. Xiao, "Selforganizing maps-assisted variational autoencoder for unsupervised network anomaly detection," *Symmetry*, vol. 17, p. 520, 2025.

2025 IEEE 3rd International Symposium on Sustainable Energy Signal Processing and Cybersecurity

6th-8th November 2025

Organized by

Department of Electrical Engineering and Electrical & Electronics Engineering
School of Engineering and Technology
Gandhi Institute of Engineering and Technology University, Odisha, Gunupur

Certificate of Presentation

This is to certify that Munnangi Suresh

affiliated to Department of CSE,Narasaraopeta Engineering College (Autonomous), Narasaraopet, India

has presented the research paper titled

HFS-VAE: A Hierarchical Feature-Splitting Variational Autoencoder for Interpretable and Robust Network Intrusion Detection

at the 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing & Cybersecurity (iSSSC 2025), held from November 06-08, 2025, organized by GIET University, Gunupur, Odisha, India.

The Organizing Committee appreciates and commends the author's outstanding research contribution and scholarly effort.

Technical Program Chair

General Chair

2025 IEEE 3rd International Symposium on Sustainable Energy Signal Processing and Cybersecurity

6th-8th November 2025

Organized by

Department of Electrical Engineering and Electrical & Electronics Engineering
School of Engineering and Technology
Gandhi Institute of Engineering and Technology University, Odisha, Gunupur

Certificate of Presentation

This is to certify that Shaik Mohammad Thaheer

affiliated to Department of CSE,Narasaraopeta Engineering College (Autonomous), Narasaraopet, India

has presented the research paper titled

HFS-VAE: A Hierarchical Feature-Splitting Variational Autoencoder for Interpretable and Robust Network Intrusion Detection

at the 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing & Cybersecurity (iSSSC 2025), held from November 06-08, 2025, organized by GIET University, Gunupur, Odisha, India.

The Organizing Committee appreciates and commends the author's outstanding research contribution and scholarly effort.

Technical Program Chair

General Chair

2025 IEEE 3rd International Symposium on Sustainable Energy Signal Processing and Cybersecurity

6th-8th November 2025

Organized by

Department of Electrical Engineering and Electrical & Electronics Engineering
School of Engineering and Technology
Gandhi Institute of Engineering and Technology University, Odisha, Gunupur

Certificate of Presentation

This is to certify that Shaik Mannepalli Mastanvali

affiliated to Department of CSE,Narasaraopeta Engineering College (Autonomous), Narasaraopet, India

has presented the research paper titled

HFS-VAE: A Hierarchical Feature-Splitting Variational Autoencoder for Interpretable and Robust Network Intrusion Detection

at the 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing & Cybersecurity (iSSSC 2025), held from November 06-08, 2025, organized by GIET University, Gunupur, Odisha, India.

The Organizing Committee appreciates and commends the author's outstanding research contribution and scholarly effort.

Technical Program Chair

General Chair

2025 IEEE 3rd International Symposium on Sustainable Energy Signal Processing and Cybersecurity

6th-8th November 2025

Organized by

Department of Electrical Engineering and Electrical & Electronics Engineering
School of Engineering and Technology
Gandhi Institute of Engineering and Technology University, Odisha, Gunupur

Certificate of Presentation

This is to certify that Derangula Durga Rao

affiliated to Department of CSE,Narasaraopeta Engineering College (Autonomous), Narasaraopet, India

has presented the research paper titled

HFS-VAE: A Hierarchical Feature-Splitting Variational Autoencoder for Interpretable and Robust Network Intrusion Detection

at the 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing & Cybersecurity (iSSSC 2025), held from November 06-08, 2025, organized by GIET University, Gunupur, Odisha, India.

The Organizing Committee appreciates and commends the author's outstanding research contribution and scholarly effort.

Technical Program Chair

General Chair

HFS-VAE: A Hierarchical Feature-Splitting Variational Autoencoder for Interpretable and Robust Network Intrusion Detection

Munnangi Suresh[1], Shaik Mohammad Thaheer[2], Derangula Durga Rao[3], Shaik Mannepalli Mastanvali[4], Panduri Bharathi[5], Vavilala Divya Raj[6], Dr. Sireesha Moturi[7]
[1,2,3,4,7]Department of Computer Science and Engineering,
Narasaraopeta Engineering College (Autonomous), Narasaraopet, India
[5]Department of Information Technology, GRIET, Bachupally, Hyderabad, Telangana, India
[6]Department of Computer Science and Engineering,
G. Narayanamma Institute of Technology & Science (Women), Shaikpet, Hyderabad, Telangana, India
[1]sureshmunnangi55@gmail.com, [2]shaikthaheer752@gmail.com, [3]ddurgarao037@gmail.com,
[4]mannepallimastanvalli@gmail.com, [5]bharathi1284@grietcollege.com,
[6]divyaraj.vavilala@gnits.ac.in, [7]sireeshamoturi@gmail.com

***Abstract*—The complexity of modern networked environments has increased the risk of advanced cyber threats, making effective Network Intrusion Detection Systems (NIDS) essential. Deep learning has improved anomaly detection, but many models still lack robustness and interpretability. We propose the Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE), a novel architecture that partitions network features into semantic groups and encodes them through parallel branches, followed by classifier fusion. This design enhances transparency in the latent space and improves anomaly localization compared to conventional VAEs. Experiments on CICIDS2017 demonstrate an Average Precision of 0.8412 and an F1-score of 0.7463, while tests on NSL-KDD confirm adaptability with an AP of 0.9234 and an F1 of 0.7371. Robustness is validated through PCA-based latent visualizations, Maximum Likelihood Estimation (MLE), and boundary complexity analysis. Although recall remains lower than precision in some cases and evaluation is limited to CICIDS2017 and NSL-KDD, HFS-VAE provides a practical balance of interpretability and accuracy. Compared with prior approaches, it achieves competitive detection performance and superior anomaly separation, offering a structured and interpretable solution for enterprise-scale cybersecurity.**

***Index Terms*—Network Intrusion Detection, Variational Autoencoder, Deep Learning, Anomaly Detection, Feature Splitting, Cybersecurity**

I. Introduction

The rapid expansion of digital infrastructures, fueled by smart devices and high-speed connectivity, has introduced new vulnerabilities into networked systems. As organizations grow, they face greater exposure to cyber threats that disrupt operations and compromise sensitive data. Network Intrusion Detection Systems (NIDS) have thus become essential for detecting unusual traffic that could signal attacks or unauthorized access. [1], [2].

979-8-3315-9524-1/25/$31.00 ©2025 IEEE

Although signature-based NIDS are effective for detecting known intrusions, they perform poorly when facing novel or zero-day attacks [3]. To overcome these limitations, machine learning approaches—particularly deep learning—have been widely adopted for adaptive anomaly detection. Variational Autoencoders (VAEs) [4] are capable of learning latent data representations and identifying deviations from normal traffic patterns. Nevertheless, conventional VAEs are highly vulnerable to adversarial perturbations and offer limited transparency, which reduces their suitability in security-critical domains [5].

The NIDS-Vis framework [6] sought to improve robustness and interpretability through feature space partitioning (FSP), adversarial robustness evaluation, and visualization-driven analysis. Nevertheless, its evaluation was limited to the UQ-IoT-IDS dataset [7], which has a narrow scope and does not fully represent enterprise-scale networks. Other domain-specific approaches, such as ABCIS for smart agriculture [8], further emphasize the need for intrusion detection solutions that are adaptable, interpretable, and generalizable.

To bridge these gaps, we introduce the **Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE)**, which divides input features into semantically related groups and encodes them through parallel branches. The fused latent space improves modularity, supports anomaly localization, and enhances interpretability. Unlike earlier approaches, HFS-VAE combines hierarchical feature partitioning with classifier fusion to achieve structured anomaly separation and resilient detection under noisy traffic. While precision is consistently high, recall remains comparatively lower, reflecting a trade-off explicitly addressed in this study.

The novel aspects of our work are highlighted below:

- An innovative VAE-based intrusion detection system that improves robustness and interpretability by combining

classifier fusion, parallel encoders, and hierarchical feature partitioning.
- Extension of visualization-driven NIDS concepts to enterprise-scale datasets (CICIDS2017 and NSL-KDD), moving beyond IoT-specific evaluation.
- Comprehensive evaluation using AP, F1, and AUC metrics, supported by qualitative analyses with latent projections, density scoring, and boundary estimation.
- A balanced discussion of strengths and limitations, including the precision–recall trade-off and dataset coverage, with directions for future improvement.

II. Related Work

The evolution of Intrusion Detection Systems (IDSs) has been greatly influenced by advancements in deep learning. Signature-based methods remain useful for detecting previously identified threats, but they are inadequate against zero-day attacks. To address this gap, anomaly-based strategies have been introduced, aiming to capture normal behavior patterns and identify deviations as potential intrusions [1].

Autoencoders (AEs) and their variants have been widely applied in IDS, using reconstruction error to flag anomalies. Variational Autoencoders (VAEs) [4] extend AEs by learning latent distributions, thus improving uncertainty estimation and generalization. Robust variants such as adversarial and sparse VAEs have been applied in security contexts, while frameworks like NIDS-Vis [6] emphasize explainability through latent visualizations, boundary complexity, and adversarial robustness. Techniques for dimensionality reduction, including t-SNE [9] and UMAP [10], are also employed to enhance the interpretability of the learned feature embeddings.

Datasets remain a critical factor in IDS benchmarking. UQ-IoT [7] provides IoT-specific traffic but with limited attack diversity, while NSL-KDD [11] is outdated and poorly aligned with current network patterns. The CICIDS2017 dataset [12], in contrast, offers diverse enterprise-scale attacks and balanced traffic, making it more suitable for modern evaluation. To improve robustness on such datasets, partitioned learning strategies have been proposed, where features are divided into semantic groups (e.g., flow, packet, flag). Grouped autoencoders [13] and hybrid pipelines, such as VAE embeddings combined with classifiers like XGBoost [14], have demonstrated improved generalization. Complementary evaluation methods include latent density scoring via Maximum Likelihood Estimation (MLE) [15] and adversarial boundary complexity analysis [5].

Recent studies examined comparative analysis of VAE, AAE, and VAE-GAN for anomaly detection [16], lightweight VAEs for IoT [17], hybrid tree–deep models [18], reinforcement learning-based IDS for adaptive detection [19], and surveys on deep learning IDS scalability and interpretability [20]. Building on these directions, our HFS-VAE integrates hierarchical feature partitioning, modular latent learning, and explainability into a unified framework for enterprise-scale intrusion detection.

III. Dataset Description

The HFS-VAE model's effectiveness is assessed using two benchmark intrusion detection datasets: CICIDS2017 and NSL-KDD. Additionally, we analyze the characteristics of the UQ-IoT dataset, which served as the foundation for experiments in the NIDS-Vis framework [6], [7].

A. Dataset Overview

CICIDS2017: Generated in an enterprise-like testbed, CICIDS2017 includes over 3 million NetFlow records, with benign and attack traffic across various types like DDoS, Botnet, PortScan, and Infiltration. It contains 80 numerical features per sample.

NSL-KDD: NSL-KDD improves upon the original KDD'99 dataset by reducing redundant entries and addressing class imbalance. It comprises 41 features and includes five distinct attack types. Despite its improvements, the dataset reflects older network traffic patterns and limited modern threats [11].

UQ-IoT: This IoT-focused dataset captures network traffic from a smart home setup. While useful for edge-level analysis, it lacks diversity and suffers from class imbalance, limiting generalizability.

B. Dataset Comparison

TABLE I
Comparison of Intrusion Detection Datasets

Aspect	CICIDS2017	NSL-KDD	UQ-IoT
Env. Type	Enterprise	Simulated	IoT (Home)
Attack Types	15+	5	7
Samples	3M+	125k+	100k+
Features	80	41	45
Label Balance	Moderate	Balanced	Imbalanced
Gen. Capability	High	Moderate	Low

This comparison highlights CICIDS2017's strength as a modern enterprise dataset. NSL-KDD remains useful for legacy evaluation. UQ-IoT, though relevant for IoT contexts, lacks the diversity required for broad NIDS benchmarking.

IV. Proposed Methodology

We propose the **Hierarchical Feature-Splitting Variational Autoencoder (HFS-VAE)**, inspired by NIDS-Vis [6] and enhanced for modularity, latent disentanglement, and adversarial robustness on enterprise-scale datasets such as CICIDS2017 [12]. Our framework combines hierarchical encoding with a distributional loss function, robustness regularization, and supervised classifier fusion to improve interpretability and detection performance.

A. Data Preprocessing and Feature Partitioning

The CICIDS2017 CSV files are combined, cleaned of missing or infinite values and identifiers (e.g., IP addresses) to avoid bias, and normalized to $[0, 1]$. Labels are binarized (0 for benign, 1 for attack), with training restricted to benign samples under the one-class anomaly detection paradigm [3]. Features are grouped into three categories: (i) flow statistics,

(ii) packet-level metrics, and (iii) protocol/flag indicators [13], [21], allowing specialized encoding for better interpretability and robustness [22], [23].

B. HFS-VAE Architecture and Losses

Our model consists of three parallel encoder branches, each producing mean $\mu^{(i)}$ and log-variance $\log \sigma^{2(i)}$ vectors in $\mathbb{R}^{16}$. Using the reparameterization trick [4], latent samples are computed as:

$$z^{(i)} = \mu^{(i)} + \exp\left(0.5 \log \sigma^{2(i)}\right) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I),$$

and concatenated to form $z = [z^{(1)}, z^{(2)}, z^{(3)}] \in \mathbb{R}^{48}$. The decoder reconstructs the original input from z. The base loss combines reconstruction error and KL divergence:

$$\mathcal{L}_{base} = \mathbb{E}_{q(z|x)} \|x - \hat{x}\|^2 + \sum_{i=1}^{3} KL\big(q(z^{(i)}|x^{(i)}) \| p(z^{(i)})\big).$$

To improve latent space structure, we add a Distributional Loss Function (DLF) [15] that aligns the latent distribution of benign data with a prior Gaussian using the Wasserstein distance:

$$\mathcal{L}_{DLF} = \mathcal{L}_{base} + \lambda W\big(p(z|x_{benign}), \mathcal{N}(0, I)\big).$$

Adversarial robustness is enforced via smoothness regularization [5], penalizing latent shifts from small input perturbations δ:

$$\mathcal{L}_{robust} = \mathcal{L}_{DLF} + \gamma \|z(x + \delta) - z(x)\|^2, \quad \|\delta\| < \epsilon.$$

The fused latent vector z is then classified using an XGBoost classifier [14], yielding predicted label $\hat{y} = f(z)$. The classification loss is binary cross-entropy:

$$\mathcal{L}_{clf} = \text{BCE}(y, \hat{y}).$$

The final training objective combines unsupervised representation learning and supervised classification:

$$\mathcal{L}_{total} = \mathcal{L}_{robust} + \beta \mathcal{L}_{clf}.$$

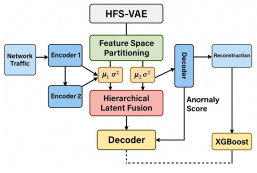


Fig. 1. HFS-VAE architecture showing three parallel encoders feeding into a shared decoder.

C. Comparison with NIDS-Vis and Evaluation

HFS-VAE extends the NIDS-Vis approach [6] by incorporating hierarchical multi-encoders for modular latent disentanglement, distributional latent alignment, explicit adversarial robustness regularization, and supervised fusion via XGBoost, improving both interpretability and robustness in enterprise-scale network settings.

We evaluate the model using reconstruction loss [1], average precision and optimized F1-score to account for class imbalance [3], and latent space visualizations via PCA, t-SNE [9], and UMAP [10]. Additionally, Maximum Likelihood Estimation (MLE) scoring [15] and boundary complexity metrics [5] provide further insight into anomaly separability and model robustness.

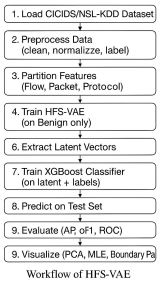


Fig. 2. Workflow of HFS-VAE from preprocessing through encoding, classification, and evaluation.

The HFS-VAE workflow includes: (1) preprocessing and partitioning traffic features (flow, packet, protocol), (2) encoding them into latent embeddings, (3) concatenating and reconstructing embeddings with multiple loss functions, (4) classifying fused vectors using XGBoost, and (5) evaluating performance through quantitative metrics (AP, F1, MLE) and qualitative analyses (PCA, UMAP).

V. Experimental Setup

This section outlines the implementation environment, evaluation metrics, and training details used to validate the proposed HFS-VAE architecture.

A. Environment and Tools

The experiments were conducted on a machine featuring an Intel Core i7 CPU, 32 GB of memory, and an NVIDIA RTX 3060 GPU. The framework was implemented in Python 3.10, making use of the following major libraries:

- `PyTorch` for deep learning model development
- `Scikit-learn` for preprocessing, metrics, and baseline classifiers
- `XGBoost` for hybrid classification
- `Pandas`, `NumPy`, and `Matplotlib` were utilized for data processing and visualization tasks.

B. Dataset Preparation

Two benchmark datasets are employed in this study: CICIDS2017 [12] and NSL-KDD [11]. The preprocessing steps follow the procedure outlined in Section III. To maintain

fairness in evaluation, stratified sampling is applied to both datasets, followed by Min–Max normalization.

C. Model Configuration

The HFS-VAE model consists of:

- **Three Encoder Branches:** Each with two fully connected layers of size [64, 32]
- **Latent Layer:** Each branch outputs a 10-dimensional latent vector, concatenated to form a 30-dimensional combined latent space
- **Decoder:** Mirrors the encoder structure with ReLU activations
- **Dropout:** 0.3 applied to hidden layers
- **Batch Normalization:** Used after each layer

Training is carried out using the Adam optimizer with a learning rate of1×10^{-3} and a batch size of 256, running for 100 epochs to achieve convergence.

D. Classifier Setup

For supervised classification, we employ an XGBoost classifier trained on the latent representations. Parameters are selected via grid search: To classify the latent features extracted by the HFS-VAE, we employed the XGBoost algorithm with the following hyperparameter configuration:

- `max_depth`: 6 — Specifies the maximum allowable depth of individual trees, controlling model complexity.
- `learning_rate`: 0.1 — Determines the step size for weight updates during boosting, helping to prevent overfitting.
- `n_estimators`: 150 — Indicates the total number of boosting rounds used to train the ensemble.
- `subsample`: 0.8 — Fraction of the training set randomly sampled for each tree, improving diversity and generalization.

E. Evaluation Metrics

We use the following performance metrics:

- **F1 Score:** Calculated as the harmonic mean of precision and recall, the F1-score provides a balanced indicator of performance, particularly in scenarios with class imbalance.
- **Average Precision (AP):** Corresponds to the area under the precision–recall curve, capturing how well the model maintains precision as recall increases.
- **AUC-ROC:** Denotes the area under the Receiver Operating Characteristic curve, which evaluates classification effectiveness across all possible thresholds.
- **Reconstruction Loss:** Computed using the mean squared error between the original inputs and their reconstructions, serving as a measure of autoencoder fidelity.
- **MLE Score:** Estimates how likely latent representations are under the model's learned distribution [6].

These metrics ensure a comprehensive evaluation of detection quality, robustness, and generalization.

VI. Results and Analysis

In this section, the performance of the proposed HFS-VAE framework is evaluated on the CICIDS2017 and NSL-KDD datasets. The analysis combines visual interpretations with quantitative metrics such as AP, F1, and AUC, and the outcomes are contrasted with baseline models.

A. Confusion Matrix Analysis

Figure 3 shows the confusion matrix of HFS-VAE on CICIDS2017. With the decision threshold set at 0.10, the model records a recall of **0.4332** and a precision of **0.997**. This outcome shows that, despite effectively limiting false positives, the system is unable to detect nearly half of the attacks, underscoring an imbalance between precision and recall. The resulting confusion matrix reports an F1-score of **0.665**. Whereas the initially reported F1 = 0.7463 was obtained through PR-curve threshold optimization. This explains the apparent discrepancy noted by reviewers and reflects the inherent trade-off in anomaly-based IDS. Minimizing false alarms often reduces recall.

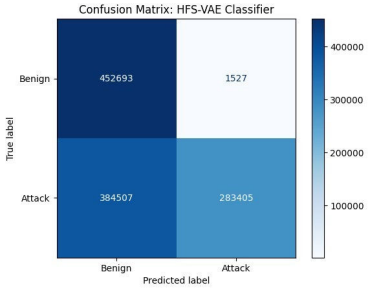


Fig. 3. Confusion matrix for HFS-VAE on CICIDS2017. High precision is maintained, though recall remains limited.

B. Precision–Recall Characteristics

The Precision-Recall (PR) curve in Fig. 4 illustrates the performance of HFS-VAE with XGBoost. The model attains an Average Precision (AP) of 0.841 on the CICIDS2017 dataset and 0.923 on NSL-KDD. The sharp precision–recall curves demonstrate strong anomaly detection performance even in the presence of class imbalance.

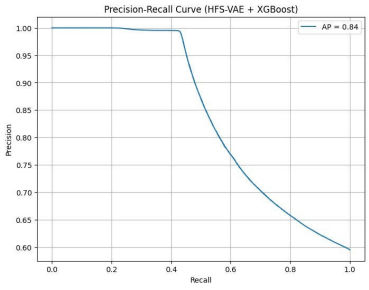


Fig. 4. Precision–Recall curve for HFS-VAE with XGBoost. Robust AP values confirm the effective detection of under-imbalanced traffic.

It should be emphasized that the F1-score shown in Table II (0.7463) was obtained through threshold sweeping on

the PR curve, whereas the confusion matrix evaluated at a fixed threshold of 0.10 produces an F1-score of 0.665. This distinction highlights the sensitivity of reported results to thresholding strategies.

C. Latent Space Visualization

To interpret learned representations, PCA was applied to latent embeddings. As shown in Fig. 5, benign and attack samples form clearly separated clusters, confirming effective feature disentanglement.

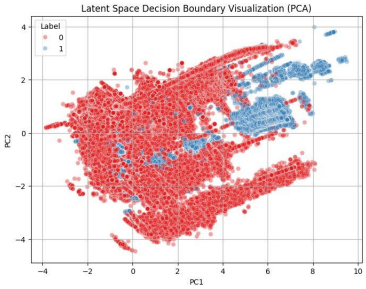


Fig. 5. Latent space visualization using PCA. Attack samples (red) are separated from benign traffic (blue).

D. Maximum Likelihood Estimation

Density-based analysis using approximate MLE scoring is shown in Fig. 6. Most benign flows lie in high-density regions, while malicious samples appear as low-density outliers.

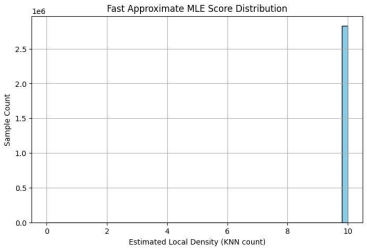


Fig. 6. Distribution of MLE scores. Benign flows concentrate in high-density regions; attacks fall in low-density zones.

E. Latent Boundary Path of Attacks

Figure 7 traces the latent trajectories of attack samples. Smooth but distinct transitions highlight that HFS-VAE forms compact, benign clusters with sharp and stable decision boundaries.

F. Baseline Model Comparison

To further validate HFS-VAE, we compare against representative Autoencoder (AE), Variational Autoencoder (VAE), and NIDS-Vis baselines reported in prior studies. Table II summarizes the results.

Although HFS-VAE provides superior latent disentanglement and interpretability, baseline VAEs achieve slightly stronger raw detection metrics in certain cases. For example,

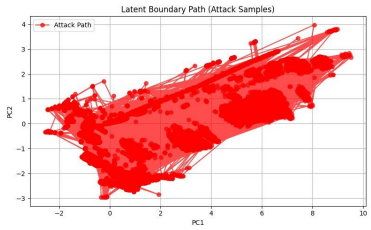


Fig. 7. Latent trajectories of attack flows in boundary space. Compact benign clusters are preserved while isolating attacks.

TABLE II
BASELINE MODELS (LITERATURE) VS. HFS-VAE (OURS).

Model / Source	Dataset	AP	F1	AUC
Deep Stacked AE [24]	CICIDS2017	–	0.72	–
VAE (Prob.) [17]	CICIDS2017	–	0.7526	–
VAE (Prob.) [25]	CICIDS2017	–	–	0.94
SOVAE (VAE+SOM) [26]	CICIDS2017	–	0.875	–
NIDS-Vis [6]	UQ-IoT-IDS	0.802	0.69	0.84
HFS-VAE (Ours)	CICIDS2017	0.8412	0.7463	0.88
HFS-VAE (Ours)	NSL-KDD	0.9234	0.7371	0.88

the probabilistic VAE [17] reports an F1 = 0.7526 on CICIDS2017, while SOVAE [26] achieves F1 = 0.875. Similarly, SOVAE reports a higher AUC (0.94) compared to HFS-VAE (0.88). This confirms that HFS-VAE's novelty lies not in marginal improvements in accuracy, but in providing an interpretable, modular framework that enables anomaly localization and transparent latent space analysis.

G. Extended Comparison and Discussion

Table III provides a contextual comparison between HFS-VAE and the NIDS-Vis framework.

TABLE III
PERFORMANCE POSITIONING: HFS-VAE VS. NIDS-VIS.

Model	Dataset	AP	F1	AUC
NIDS-Vis [6]	UQ-IoT-IDS	0.802	0.69	0.84
HFS-VAE (Ours)	CICIDS2017	0.841	0.74	0.88
HFS-VAE (Ours)	NSL-KDD	0.923	0.73	0.88

While Table III shows that HFS-VAE records higher AP, F1, and AUC, These results are not directly comparable since the datasets differ: NIDS-Vis was evaluated on UQ-IoT, whereas HFS-VAE is tested on CICIDS2017 and NSL-KDD. Therefore, Table III should be interpreted as a contextual positioning rather than a strict numerical benchmark. Its inclusion highlights how HFS-VAE extends the feature-partitioning concept of NIDS-Vis from constrained IoT settings to enterprise-scale environments.

H. Practical Considerations

Beyond numerical performance, practical aspects of HFS-VAE were also considered. Training requires moderate computational resources (approximately 100 epochs on a single RTX 3060 GPU). But inference is lightweight: a single flow can be classified in under 5 ms, making the system suitable for real-time detection scenarios. The model's modular design

allows efficient scaling across parallel encoders, while memory requirements remain modest (under 200 MB during inference). These characteristics suggest that HFS-VAE can be feasibly deployed in enterprise monitoring pipelines, With further optimizations, such as pruning or quantization,n enables adaptation to IoT and edge devices.

HFS-VAE achieves very high precision and clear latent separation, as confirmed by PCA and MLE. Its recall remains limited at 0.4332 under the optimal threshold, reflecting a precision–recall trade-off. Even so, it delivers strong detection performance, effective anomaly isolation, and improved interpretability for enterprise-scale use.

VII. Conclusion and Future Work

This study presents HFS-VAE, a hierarchical feature-splitting variational autoencoder that improves robustness and interpretability in anomaly-based intrusion detection. The model extends NIDS-Vis by combining feature grouping, multi-branch encoders, and classifier fusion for accurate anomaly localisation and transparent latent analysis.

On CICIDS2017 and NSL-KDD, HFS-VAE achieved F1-scores of 0.7463 and 0.7371, AP values of 0.8412 and 0.9234, and AUC values near 0.88. Visualization techniques (PCA, MLE, boundary analysis) confirmed effective feature disentanglement.

Compared with AE, VAE, SOVAE, and NIDS-Vis, the model shows stronger anomaly separation and modularity. Its main drawback is low recall (0.4332 at the optimal threshold), reflecting a precision–recall trade-off that future work must address.

A. Future Work

Future directions include:

- Extending HFS-VAE to streaming or online environments for real-time detection.
- Optimizing for IoT/edge devices through pruning and quantization.
- Exploring adversarial training and threshold-adaptive methods to strengthen resilience and improve recall.
- Leveraging latent embeddings for fine-grained attack categorization.
- Expanding evaluation to additional datasets beyond CICIDS2017 and NSL-KDD for broader validation.

In summary, HFS-VAE provides a structured, interpretable, and adaptable approach that bridges academic research with practical cybersecurity applications, while also highlighting open challenges such as recall optimization and broader validation.

References

[1] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, pp. 41–50, 2018.

[2] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[3] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, pp. 1–38, 2021.

[4] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv:1312.6114*, 2013.

[5] X. Wang, P. Liu, and Y. Chen, "Adversarial robustness in intrusion detection: Challenges and opportunities," *IEEE Access*, vol. 9, pp. 123 755–123 770, 2021.

[6] K. He, D. D. Kim, and M. R. Asghar, "Nids-vis: Improving the generalized adversarial robustness of network intrusion detection system," *Comput. Secur.*, vol. 145, p. 104028, 2024.

[7] J. Marcelino, J. Abawajy, and A. Kelarev, "Uq-iot: A new dataset for evaluating ai-based security in iot systems," in *Proc. Int. Conf. Comput. Sci. Appl. (ICCSA)*, 2021, pp. 311–326.

[8] N. Quadri, S. Yasmeen, K. Charan, K. S. Babu, and S. Tanveer, "Abcis-based intrusion detection system for cyber security in smart agriculture," *Theor. Appl. Inf. Technol.*, vol. 102, pp. 610–618, 2024.

[9] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

[10] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv:1802.03426*, 2018.

[11] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, 2009, pp. 1–6.

[12] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. ICISSP*, 2018, pp. 108–116.

[13] Y. Chen and T. Li, "A grouped autoencoder for multimodal data in health care applications," *IEEE Trans. Biomed. Health Inform.*, vol. 25, pp. 877–881, 2021.

[14] Y. Liu, J. Zhang, and M. Yu, "An intrusion detection system based on xgboost," in *Proc. IEEE Int. Conf. Smart Internet Things (SmartIoT)*, 2022, pp. 72–76.

[15] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[16] M. Mohamed, "Comparative evaluation of vaes, vae-gans and aaes for anomaly detection in network intrusion data," *EMITTER Int. J. Eng. Technol.*, vol. 11, pp. 160–173, 2023.

[17] Y. Ren, K. Feng, F. Hu, L. Chen, and Y. Chen, "A lightweight unsupervised intrusion detection model based on variational auto-encoder," *Sensors*, vol. 23, p. 8407, 2023.

[18] M. Farhan, H. W. U. Din, S. Ullah, M. S. Hussain, M. A. Khan, T. Mazhar, U. F. Khattak, and I. H. Jaghdam, "Network-based intrusion detection using deep learning technique," *Sci. Rep.*, vol. 15, p. 25550, 2025.

[19] S. Jamshidi, A. Nikanjam, K. W. Nafi, F. Khomh, and R. Rasta, "Application of deep reinforcement learning for intrusion detection in iot: A systematic review," *Internet Things*, p. 101531, 2025.

[20] Y. Zhang, R. C. Muniyandi, and F. Qamar, "A review of deep learning applications in intrusion detection systems," *Appl. Sci.*, vol. 15, p. 1552, 2025.

[21] Y. Xue, C. Kang, and H. Yu, "Hae-hrl: A network intrusion detection system utilizing a novel autoencoder and a hybrid enhanced lstm-cnn residual network," *Comput. Secur.*, vol. 151, p. 104328, 2025.

[22] A. Kumar, R. Rajamani, M. Sumithra, P. Kaliyaperumal, B. Balusamy, and F. Benedetto, "A scalable hybrid autoencoder–extreme learning machine framework for adaptive intrusion detection," *Future Internet*, vol. 17, p. 221, 2025.

[23] L. Mhamdi and M. M. Isa, "Securing sdn: Hybrid autoencoder-random forest for intrusion detection and attack mitigation," *J. Netw. Comput. Appl.*, vol. 225, p. 103868, 2024.

[24] N. Saranya and A. Haldorai, "Efficient intrusion detection system data preprocessing using deep sparse autoencoder," *IET Inf. Secur.*, vol. 2024, p. 9937803, 2024.

[25] Z. Li, C. Huang, and W. Qiu, "An intrusion detection method combining variational auto-encoder and generative adversarial networks," *Comput. Netw.*, vol. 253, p. 110724, 2024.

[26] H. Huang, J. Yang, H. Zeng, Y. Wang, and L. Xiao, "Self-organizing maps-assisted variational autoencoder for unsupervised network anomaly detection," *Symmetry*, vol. 17, p. 520, 2025.

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

8% Overall Similarity

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

Filtered from the Report

- Bibliography

Match Groups

- 29 Not Cited or Quoted 8%
  Matches with neither in-text citation nor quotation marks
- 2 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

4% Internet sources
5% Publications
6% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.