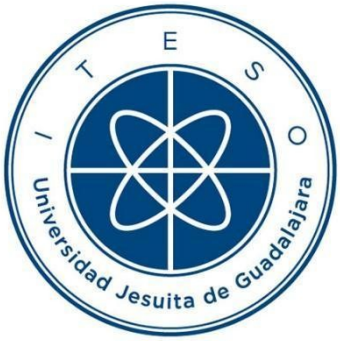
INSTITUTO TECNOLÓGICO
Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

DOCTORADO EN CIENCIAS DE LA INGENIERÍA



DETECCIÓN DE ANOMALÍAS EN TRÁFICO DE RED MEDIANTE LA INTEGRACIÓN DE PERFILADO CONDUCTUAL DE USUARIOS Y APRENDIZAJE PROFUNDO

Tesis que para obtener el grado de

DOCTOR EN CIENCIAS DE LA INGENIERÍA

presenta: Cristian Salvador Najar Juárez

Director de tesis: Dr. Luis Julián Domínguez Pérez

Tlaquepaque, Jalisco. Noviembre de 2026

DOCTOR EN CIENCIAS DE LA INGENIERÍA (2022) ITESO, Tlaquepaque, Jal., México

TÍTULO: **Detección de anomalías en tráfico de red mediante la integración de perfilado conductual de usuarios y aprendizaje profundo**

AUTOR: Cristian Salvador Najar Juárez
Ingeniero en Software (Universidad Autónoma de Guadalajara, México)
Maestro en Tecnologías de la Información (Universidad Autónoma de Guadalajara, México)

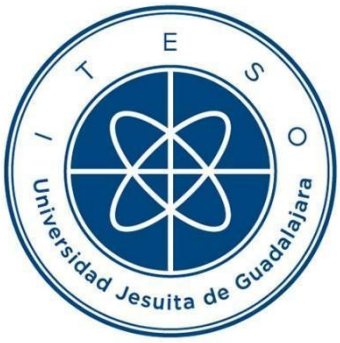
DIRECTOR DE TESIS: Luis Julián Domínguez Pérez
Departamento de Electrónica, Sistemas e Informática, ITESO
Ingeniero en Electrónica (ITESO, México)
Maestro en Sistemas Electrónicos (ITESM Campus Monterrey, México)
Doctor en Ingeniería Eléctrica (Universidad McMaster, Canadá)
Senior, IEEE

NÚMERO DE PÁGINAS: 136.

ITESO – The Jesuit University of Guadalajara

Department of Electronics, Systems, and Informatics

DOCTORAL PROGRAM IN ENGINEERING SCIENCES



DETECTION OF ANOMALIES IN NETWORK TRAFFIC THROUGH THE INTEGRATION OF USER BEHAVIORAL PROFILING AND DEEP LEARNING

Thesis to obtain the degree of

DOCTOR IN ENGINEERING SCIENCES

Presented by: Cristian Salvador Najar Juárez

Thesis Director: Dr. Luis Julián Domínguez Pérez

Tlaquepaque, Jalisco, Mexico
November 2026

PHD IN ENGINEERING SCIENCES (2022) ITESO, Tlaquepaque, Jal., Mexico

TITLE: **Detection of anomalies in network traffic through the integration of user behavioral profiling and deep learning**

AUTHOR: Cristian Salvador Najar Juárez
Bachelor's degree in software engineering (Universidad Autónoma de Guadalajara, Mexico)
Master's degree in information technologies (Universidad Autónoma de Guadalajara, Mexico)

THESIS DIRECTOR: Luis Julián Domínguez Pérez
Department of Electronics, Systems, and Informatics, ITESO
Bachelor's degree in electronics engineering (ITESO, Mexico)
Master's degree in electronic systems (ITESM Campus Monterrey, Mexico)
Ph.D. degree in electrical engineering (McMaster University, Canada)
Senior, IEEE

NUMBER OF PAGES: 136.

I dedicate this doctoral thesis to God for His guidance and strength throughout this journey. To my beloved wife, for her unwavering support, love, and patience; to my daughter, who inspires me every day; and to my family, whose encouragement and faith in me made this achievement possible. This accomplishment is as much yours as it is mine.

Resumen

Las anomalías del tráfico de red a menudo se manifiestan como desviaciones en los patrones de tráfico y, en muchos casos, están asociadas con desviaciones de los perfiles típicos de comportamiento del usuario. Puesto que la detección de anomalías de red se orienta a distinguir actividad legítima de actividad que indica riesgo, el perfilado de usuarios es de utilidad ya que permite describir hábitos de interacción y construir una referencia conductual que aporta contexto al análisis del tráfico. Esta relación entre patrones de conexión y patrones de uso sustenta la protección de servicios, datos y procesos en infraestructuras digitales, ya que permite identificar desviaciones asociadas con accesos no autorizados, uso indebido de recursos y otras amenazas que afectan la operación de la red.

Los enfoques tradicionales enfrentan el problema de detectar estas desviaciones mediante reglas, umbrales estadísticos y modelos de aprendizaje automático aplicados al tráfico, pero su alcance se amplía cuando el análisis incorpora información de comportamiento asociada al usuario y no se limita al tráfico aislado.

Esta tesis evalúa distintos enfoques de detección de anomalías donde el perfilado de usuarios forma parte de la estrategia de análisis y se integra con el estudio del tráfico de red en un marco común. El proceso incluye la captura de actividad web, la construcción de vectores de comportamiento, la generación de representaciones latentes y el uso de técnicas de aprendizaje profundo para relacionar comportamiento y tráfico dentro de un mismo espacio de características. Se comparan métodos basados en similitud, agrupamiento, clasificación supervisada y fusión de representaciones, y se examina su desempeño bajo criterios homogéneos de evaluación.

Entre los enfoques analizados, el agrupamiento híbrido basado en autoencoder alcanza la mayor calidad de perfiles de comportamiento, con un coeficiente de silueta (silhouette) de hasta 0.8632 y un índice Davies–Bouldin de 0.5775. Sin embargo, aplicado sobre la totalidad de los conjuntos de datos, el enriquecimiento conductual no supera a un clasificador de referencia entrenado únicamente con tráfico, una vez eliminada la fuga de datos detectada en el flujo de procesamiento (pipeline): el modelo enriquecido alcanza una exactitud de 93.53% frente a 93.28% del modelo base en UNSW-NB15, y de 95.39% frente a 95.98% en CIC-IDS2017, sin una ventaja consistente para el enriquecimiento. La única condición bajo la cual el enriquecimiento produce una mejora medible es una estrategia de cascada de confianza que reserva la información

conductual para los registros de tráfico cercanos a la frontera de decisión del clasificador; en ese esquema, el AUC-ROC mejora entre 0.022 y 0.051 frente a un control de ruido en UNSW-NB15, con significancia estadística confirmada en una corrida de confirmación. Estos resultados respaldan una conclusión más acotada: la integración de perfilado conductual y tráfico de red es viable y no arbitraria, pero mejora la detección de anomalías solo bajo condiciones específicas, no como una mejora generalizada aplicable a la totalidad del tráfico.

Summary

Network traffic anomalies often manifest as deviations in traffic patterns and, in many cases, are associated with deviations from typical user behavior profiles. Since network anomaly detection aims to distinguish legitimate activity from activity that indicates risk, user profiling is useful because it allows for the description of interaction habits and the construction of a behavioral reference that provides context for traffic analysis. This relationship between connection patterns and usage patterns underpins the protection of services, data, and processes in digital infrastructures, as it allows for the identification of deviations associated with unauthorized access, misuse of resources, and other threats that affect network operation.

Traditional approaches face the challenge of detecting these deviations using rules, statistical thresholds, and machine learning models applied to traffic, but their scope expands when the analysis incorporates user-related behavioral information and is not limited to isolated traffic.

This thesis evaluates different anomaly detection approaches where user profiling is part of the analysis strategy and is integrated with the study of network traffic within a common framework. The process includes capturing web activity, constructing behavioral vectors, generating latent representations, and using deep learning techniques to relate behavior and traffic within the same feature space. Methods based on similarity, clustering, supervised classification, and representation fusion are compared, and their performance is examined under homogeneous evaluation criteria.

Among the approaches analyzed, the hybrid autoencoder-based clustering achieves the strongest behavioral profile quality, with a silhouette score of up to 0.8632 and a Davies–Bouldin index of 0.5775. However, once applied across the full datasets, behavioral enrichment does not outperform a leakage-free, traffic-only baseline classifier: the enriched model reaches 93.53%

accuracy against 93.28% for the baseline on UNSW-NB15, and 95.39% against 95.98% on CIC-IDS2017, with no consistent advantage for enrichment. The one condition under which enrichment yields a measurable gain is a confidence-cascade strategy that reserves behavioral information for traffic records near the classifier's decision boundary; there, ROC AUC improves by 0.022 to 0.051 over a noise control on UNSW-NB15, with statistical significance confirmed on a held-out confirmatory run. These results support a narrower conclusion: integrating behavioral profiling with network traffic is feasible and not arbitrary, but it improves anomaly detection only under specific conditions, not as a general improvement across the full traffic stream.

Acknowledgements

The author wishes to express his sincere appreciation to Luis Julián Domínguez Pérez, doctoral thesis director, for his encouragement, expert guidance, and dedicated supervision throughout the development of this work.

The author gratefully acknowledges the financial support provided through a scholarship granted by the Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI), Mexican Government.

The author also thanks his daughter, Lyanna Najar, and his students for their support.

Finally, special thanks are due to his family for their understanding, patience, and continuous loving support.

Contenido

Resumen 6

Summary 7

Agradecimientos 9

Contenido 10

Lista de figuras 21

Lista de tablas 23

1. Introducción 1

1.1. Motivación 4

1.2. Planteamiento del problema 5

1.3. Alcance y limitaciones 5

1.4. Objetivo general 5

1.4.1. Objetivos específicos 6

1.5. Metodología 6

1.5.1. Herramientas tecnológicas 7

1.6. Organización de la tesis 7

2. Fundamentos teóricos 9

2.1. Fundamentos de redes y datos 9

2.1.1. Modelos y protocolos de comunicación de red 9

2.1.2. Tráfico de red como dato observacional 10

2.1.3. Comportamiento del usuario en entornos de red 11

2.1.4. Representación de datos en espacios de características 12

2.2. Fundamentos de aprendizaje automático 13

2.2.1. Paradigmas de aprendizaje 14

2.2.2. Aprendizaje de representaciones y espacios latentes 15

2.2.3. Clasificación y funciones de decisión 16

2.2.4. Medidas de similitud y funciones de distancia 16

3. Perfilado conductual de usuarios para la seguridad de red 19

3.1. Introducción 19

3.2. Antecedentes del comportamiento del usuario en redes 20

3.3. Recolección de datos conductuales 22

3.4. Construcción del perfil conductual 28

3.4.1. Construcción de características conductuales 28

3.4.2. Aprendizaje de la representación conductual 31

3.4.3. Agrupamiento conductual 33

3.4.4. Conjunto de datos del perfil conductual 35

3.5. Resultados del perfilado conductual 36

3.6. Resumen del perfil conductual 38

3.7. Ética, consentimiento y manejo de datos 40

3.8. Trabajo publicado 41

4. Enriquecimiento conductual entre dominios para tráfico de red 42

4.1. Introducción 42

4.2. Antecedentes del tráfico de red y sus anomalías 43

4.3. Panorama de los conjuntos de datos de tráfico de red 46

4.4. Conjuntos de datos de tráfico de red 46

4.4.1. Conjunto de datos UNSW-NB15 46

4.4.2. Conjunto de datos CIC-IDS2017 48

4.5. Enriquecimiento conductual entre dominios 50

4.5.1. Alineación adversaria 50

4.5.2. Emparejamiento por similitud del coseno 52

4.5.3. Validez semántica del emparejamiento conductual 53

4.5.4. Construcción del conjunto de datos enriquecido 54

4.5.5. Estudio de ablación entre variantes de enriquecimiento 55

4.6. Espacio de características enriquecido 57

4.7. Resumen del marco de enriquecimiento 59

5. Modelos de aprendizaje profundo para la detección de anomalías de red 61

5.1. Introducción 61

5.2. Representación de entrada para la detección de anomalías 62

5.2.1. Representación de características del tráfico 62

5.2.2. Representación enriquecida comportamiento-tráfico 63

5.3. Arquitectura de la red neuronal 66

5.4. Estrategia de entrenamiento 67

5.5. Marco de evaluación 68

5.6. Resultados de clasificación 69

5.6.1. Resultados en el conjunto de datos UNSW-NB15 69

5.6.2. Resultados en el conjunto de datos CIC-IDS2017 76

5.6.3. Clasificadores de referencia 83

5.6.4. Enriquecimiento por cascada de confianza 83

5.6.5. Generalización entre particiones 85

5.7. Resumen del aprendizaje profundo para la detección de anomalías en redes 86

5.8. Trabajo publicado 87

6. Discusión 88

6.1. Integración de información conductual y de tráfico de red 88

6.2. Impacto del enriquecimiento conductual en la detección de anomalías 90

6.3. Implicaciones y limitaciones 91

7. Conclusiones y trabajo futuro 93

7.1. Conclusiones 93

7.2. Trabajo futuro 94

Apéndice 97

Bibliografía 119

Índice de autores 131

Índice temático 134

This page intentionally left blank.

This page intentionally left blank.

Contents

Resumen 6

Summary 7

Acknowledgements 9

Contents 17

List of Figures 21

List of Tables 23

1. Introduction 1

1.1. Motivation 4

1.2. Problem Statement 5

1.3. Scope and Limitations 5

1.4. General Objective 5

1.4.1. Specific Objectives 6

1.5. Methodology 6

1.5.1. Technological Tools 7

1.6. Thesis Organization 7

2. Theoretical Foundations 9

2.1. Network and Data Foundations 9

2.1.1. Network Communication Models and Protocols 9

2.1.2. Network Traffic as Observational Data 10

2.1.3. User Behavior in Network Environments 11

2.1.4. Data Representation in Feature Spaces 12

2.2. Machine Learning Foundations 13

2.2.1. Learning Paradigms 14

2.2.2. Representation Learning and Latent Spaces 15

2.2.3. Classification and Decision Functions 16

2.2.4. Similarity Measures and Distance Functions 16

3. User Behavioral Profiling for Network Security 19

3.1. Introduction 19

3.2. Background on User Behavior in Networks 20

3.3. Behavioral Data Collection 22

3.4. Behavioral Profile Construction 28

3.4.1. Behavioral Feature Construction 28

3.4.2. Behavioral Representation Learning 31

3.4.3. Behavioral Clustering 33

3.4.4. Behavioral Profile Dataset 35

3.5. Behavioral Profiling Results 36

3.6. Summary of Behavioral Profile 38

3.7. Ethics, Consent, and Data Handling 40

3.8. Published Work 41

4. Cross-Domain Behavioral Enrichment for Network Traffic 42

4.1. Introduction 42

4.2. Background on Network Traffic and Anomalies 43

4.3. Overview of Network Traffic Datasets 46

4.4. Network Traffic Datasets 46

4.4.1. UNSW-NB15 Dataset 46

4.4.2. CIC-IDS2017 Dataset 48

4.5. Cross-Domain Behavioral Enrichment 50

4.5.1. Adversarial Alignment 50

4.5.2. Cosine Similarity Matching 52

4.5.3. Semantic Validity of the Behavioral Match 53

4.5.4. Enriched Dataset Construction 54

4.5.5. Ablation Study Across Enrichment Variants 55

4.6. Enriched Feature Space 57

4.7. Summary of the Enrichment Framework 59

5. Deep Learning Models for Network Anomaly Detection 61

5.1. Introduction 61

5.2. Input Representation for Anomaly Detection 62

5.2.1. Traffic Feature Representation 62

5.2.2. Enriched Behavioral-Traffic Representation 63

5.3. Neural Network Architecture 66

5.4. Training Strategy 67

5.5. Evaluation Framework 68

5.6. Classification Results 69

5.6.1. Results on the UNSW-NB15 Dataset 69

5.6.2. Results on the CIC-IDS2017 Dataset 76

5.6.3. Reference Baseline Classifiers 83

5.6.4. Confidence Cascade Enrichment 83

5.6.5. Generalization Across Partitions 85

5.7. Summary of Deep Learning for Network Anomaly Detection 86

5.8. Published Work 87

6. Discussion 88

6.1. Integration of Behavioral and Network Traffic Information 88

6.2. Impact of Behavioral Enrichment on Anomaly Detection 90

6.3. Implications and Limitations 91

7. Conclusions and Future Work 93

7.1. Conclusions 93

7.2. Future Work 94

Appendix 97

Bibliography 119

Author Index 131

Subject Index 134

List of Figures

Fig. 3.1	Silhouette score vs number of clusters for K-Means clustering, Agglomerative clustering, and the hybrid model based on an autoencoder and agglomerative clustering. The results show the separation between behavioral groups used for user behavioral profiling.	36
Fig. 4.1	t-SNE visualization of embeddings from the leakage-free pipeline (split before scaling, autoencoder training, and adversarial alignment). (a) UNSW-NB15 traffic, test partition, colored by class. (b) Charles behavioral profiles. (c) UNSW-NB15 enriched. (d)-(f) same views for CIC-IDS2017.	55
Fig. 4.2	Accuracy and ROC AUC for every ablation variant on UNSW-NB15. The dashed line marks the no-enrichment baseline; every bar sits close to it.	56
Fig. 4.3	Accuracy and ROC AUC for every ablation variant on CIC-IDS2017. Nearly every enriched variant, including the real embedding, falls at or below the no-enrichment baseline; the only exception is the variant without adversarial alignment.	57
Fig. 5.1	Learning curves for DFNN training on the UNSW-NB15 dataset, from the pre-correction pipeline (kept to show training dynamics only; see Table 5.2 for the corrected metrics). (a) Model trained with enriched data shows gradual convergence with a gap between training and validation loss. (b) Model trained on traffic-only data converges within a few epochs.	70
Fig. 5.2	Confusion matrices for the DFNN on UNSW-NB15, leakage-free pipeline. (a) Baseline: 869 false positives, 1,488 false negatives. (b) Enriched: 1,368 false positives, 903 false negatives. Color intensity represents the number of samples in each cell.	71
Fig. 5.3	ROC curves for the baseline and enriched classifiers on UNSW-NB15, computed on the leakage-free test partition.	73
Fig. 5.4	Fold-level distribution of accuracy, ROC AUC, and F1-score for UNSW-NB15 under the fully retrained, leakage-free cross-validation.	75
Fig. 5.5	Learning curves of the DFNN on the CIC-IDS2017 dataset, from the pre-correction pipeline (kept to show training dynamics only; see Table 5.5 for the corrected metrics). (a) Model trained with enriched data. (b) Model trained on traffic-only data.	77
Fig. 5.6	Confusion matrices for the DFNN on CIC-IDS2017, leakage-free pipeline. (a) Baseline: 365 false positives, 45 false negatives. (b) Enriched: 414 false positives, 56 false negatives. Color intensity represents the number of samples in each cell.	78

Fig. 5.7	ROC curves for the baseline and enriched classifiers on CIC-IDS2017, computed on the leakage-free test partition.	80
Fig. 5.8	Fold-level distribution of accuracy, ROC AUC, and F1-score for CIC-IDS2017 under the fully retrained, leakage-free cross-validation.	82
Fig. 5.9	Exploratory 95% confidence interval for the ROC AUC difference between the real behavioral embedding and a noise control, confidence-cascade enrichment, medium band, UNSW-NB15, across six seeds.	84
Fig. 5.10	Seed-level ROC AUC for the real embedding against its noise control, confidence-cascade enrichment, medium band, UNSW-NB15.	85

List of Tables

Table 3.1	SERVICES ACCESSED DURING BEHAVIORAL DATA COLLECTION	25
Table 3.2	EXCERPT FROM THE BEHAVIORAL DATA COLLECTION DATASET	27
Table 3.3	AUTOENCODER ARCHITECTURE USED FOR BEHAVIORAL REPRESENTATION LEARNING	33
Table 3.4	CLUSTERING CONFIGURATION USED FOR BEHAVIORAL PROFILING	35
Table 3.5	DATASHEET BASIS FOR THE BEHAVIORAL DATASET (CHARLES PROXY LOGS)	41
Table 4.1	TRAINING PARAMETERS USED IN ADVERSARIAL AUTOENCODER DATASET ENRICHMENT	52
Table 4.2	ABLATION RESULTS ON UNSW-NB15 (SAME SPLIT, SEED, AND CLASSIFIER ACROSS VARIANTS)	56
Table 5.1	ARCHITECTURE AND TRAINING PARAMETERS OF THE DEEP FEEDFORWARD NEURAL NETWORK CLASSIFIER	65
Table 5.2	CORRECTED CLASSIFICATION METRICS ON UNSW-NB15 (LEAKAGE-FREE PIPELINE)	73
Table 5.3	MULTI-SEED COMPARISON OF BASELINE AND ENRICHED CLASSIFIERS ON UNSW-NB15 (THREE SEEDS, PAIRED T-TEST)	74
Table 5.4	CROSS-VALIDATION METRICS PER FOLD AND AVERAGE FOR THE DEEP FEEDFORWARD NEURAL NETWORK CLASSIFIER TRAINED ON THE ENRICHED UNSW-NB15 DATASET	76
Table 5.5	CORRECTED CLASSIFICATION METRICS ON CIC-IDS2017 (LEAKAGE-FREE PIPELINE)	80
Table 5.6	MULTI-SEED COMPARISON OF BASELINE AND ENRICHED CLASSIFIERS ON CIC-IDS2017 (THREE SEEDS, PAIRED T-TEST)	81
Table 5.7	CROSS-VALIDATION METRICS PER FOLD AND AVERAGE FOR THE DEEP FEEDFORWARD NEURAL NETWORK CLASSIFIER TRAINED ON THE ENRICHED CIC-IDS2017 DATASET	82

1. Introduction

The accelerated growth of the Internet, both in terms of the number of users and the volume of generated data, has significantly increased the complexity of protecting networks against security threats [Wang-Balarezo-21; Hooshmand-Hosahalli-22]. As traffic increases, so do the demands on network monitoring and management systems, which must operate on large volumes of data without degrading performance [Trevisan-Giordano-20]. In this scenario, anomaly detection becomes a key component, as it enables the identification of traffic patterns that may be associated with attacks, failures, or service disruptions [Hooshmand-Hosahalli-22].

Current traffic monitoring systems manage data flows, continuously analyze network activity, and enable responses to suspicious events. Their design depends on factors such as the available infrastructure, network architecture, security protocols, and the ability to process large information loads without loss of performance [Wang-Balarezo-21; Hooshmand-Huchaiah-24]. Within this context, multiple anomaly detection approaches have been proposed, ranging from traditional machine learning techniques to more recent methods based on deep learning, graph learning, and explainable models [Wang-Balarezo-21; Zhao-Guo-24; Aouedi-Le-25; Elsayed-Mohamed-24; Kamal-Mashaly-25; Yu-22]. These works have explored issues such as data imbalance, feature selection, model interpretability, and adaptation to IoT environments and smart networks.

Anomaly detection is particularly relevant in IoT and intelligent system environments, where the number of connected devices increases the attack surface. Early intrusion detection systems relied on statistical or signature-based methods, which were effective for known attacks but limited when facing new or adaptive threats [Hooshmand-Hosahalli-22].

To overcome these limitations, machine learning techniques were incorporated to model more complex and heterogeneous traffic behaviors, especially in homes and smart environments [Kalnoor-Gowrishankar-21; Altulaihan-Almaiah-24; Javed-Awais-24]. Likewise, lightweight and scalable approaches have been proposed to balance accuracy and efficiency, considering resource constraints on edge devices [Murthy-Asghar-24], as well as advanced architectures, including transformers, to capture long-range temporal dependencies [Wang-Yang-23]. Paradigms such as federated learning and trustworthy AI have also been explored to preserve privacy and robustness

in next-generation connected infrastructures [Garroppo-Giardina-25], along with studies focused on specific protocols such as TCP and QUIC, whose particularities influence the design of IDS models [Oran-Kocak-22; Almuhammadi-AlBakhat-22].

Despite these advances, which are mainly focused on network traffic characteristics, another relevant dimension for security has strongly emerged: user behavior profiling. This approach seeks to characterize how users interact with the network, with the aim of understanding usage patterns and strengthening protection mechanisms [Denning-87; Martin-Fernandez-21]. The construction of behavioral profiles enables the detection of activities that deviate from habitual behavior, which may indicate malicious or compromised actions [Martin-Fernandez-21; Wang-Balarezo-21].

This line of research is especially pertinent in a context where Internet usage has diversified, generating increasingly varied and dynamic traffic patterns [Trevisan-Giordano-20]. Several studies show that modeling user behavior improves threat prediction and network management by facilitating the distinction between legitimate users and attackers [Rani-Gill-23; Tao-Lu-24].

However, although network anomaly detection and user behavior profiling have been extensively studied, they have often been addressed as separate problems. The integration of user behavioral information with representations learned from network traffic remains an area under development. In this context, the present work focuses on a cross-domain data enrichment approach, where behavioral embeddings derived from users' web activity are combined with traditional network traffic features. The behavioral profiles do not correspond to the same users present in the network traffic datasets. Instead, they provide contextual behavioral patterns intended to regularize the learning process and to enrich the representation space used for anomaly detection.

Rather than replacing traffic-only methods, the proposal seeks to enrich anomaly detection datasets through an adversarial alignment process between traffic data (UNSW-NB15 and CIC-IDS2017) [Moustafa-Slay-15; Rosay-Cheval-22] and behavioral profiles obtained through proxy monitoring. This approach was first motivated by the near-perfect scores that models trained solely on traffic features appeared to reach; Chapter 5 shows that those scores came from data leakage in

an earlier version of the pipeline, so the enrichment is evaluated here against a leakage-free, traffic-only baseline.

The main contribution of this thesis lies in integrating user behavior profiling into the network anomaly detection process, and in building the experimental frame needed to test whether that integration helps, based on the premise that anomalies emerge when traffic and usage deviate from a reference behavioral profile. To this end, a common framework is proposed in which users' web activity is captured, transformed into behavioral vectors, and projected into latent representations that are related to network traffic characteristics through deep learning techniques. The behavioral dataset used in this work is intended to capture representative interaction patterns rather than the full diversity of Internet usage. Its purpose is to provide behavioral context that can enrich network traffic representations during anomaly detection.

Among the evaluated approaches, domain alignment between behavior and traffic makes it possible to place both types of information within a shared feature space, providing behavioral context to the analysis of connections and extending the scope of traditional methods focused exclusively on traffic. In this way, the model receives not only technical patterns but also deviations from usage habits, which frames anomaly detection as a joint problem of behavior and network; whether this improves detection is the question the following chapters test.

This thesis is guided by one general question and three related questions that follow from it.

The general question asks under what conditions behavioral enrichment improves anomaly detection in network traffic, and whether that effect depends on whether the enrichment reaches the whole dataset or only the cases where the traffic classifier already shows uncertainty.

Three specific questions extend the general question. On profile construction, the study asks what structure emerges from behavior recorded through Charles Proxy and whether that structure stays stable. On domain alignment, it asks whether the cosine similarity between traffic embeddings and behavioral embeddings reflects a relation that goes beyond mathematical proximity. On classifier performance, it asks whether behavioral enrichment produces a measurable gain over a baseline free of data leakage, and under which conditions that gain appears.

A testable hypothesis follows. H1: a classifier trained on traffic features enriched with

behavioral embeddings reaches a statistically significant improvement over an otherwise identical traffic-only classifier; H0: no such improvement exists. The comparison model is a classifier trained only on traffic features, sharing the same architecture, the same number of epochs, the same loss function, and the same seeds as the enriched classifier. Four metrics support the comparison: ROC AUC, precision, the Matthews correlation coefficient, and balanced accuracy, each evaluated through a paired t-test, a bootstrap of the difference, and 95% confidence intervals across multiple seeds. A result counts as relevant once it holds across seeds, survives a confirmatory run on a seed that was never used during exploration, and reaches significance under these tests.

Chapter 4 shows that this hypothesis does not hold once enrichment reaches the full dataset: replacing the behavioral embedding with zeros, Gaussian noise, or a random assignment yields results indistinguishable from the proposed method, in UNSW-NB15 and in CIC-IDS2017. Chapter 5 then reports a narrower version of the same hypothesis, limited to the cases where the traffic classifier sits near its decision boundary. That narrower version is the one the evidence supports.

1.1. Motivation

Network traffic increases with the growth of digital systems. This growth introduces risks related to attacks, misuse of resources, and unauthorized access. Systems that monitor traffic detect patterns in data flows, but these patterns do not always represent how users interact with services.

User behavior provides a complementary view of network activity. It describes how users access platforms, how they distribute their activity, and how they interact with resources over time. The integration of behavior and traffic allows a broader representation of network activity, where patterns reflect both communication and usage.

This work focuses on the relation between traffic patterns and user interaction. The motivation is to study how this relation supports the detection of anomalies in network environments.

1.2. Problem Statement

Network anomaly detection methods focus on traffic features such as packet statistics and flow characteristics. These methods detect deviations in communication patterns, but they do not include information about user activity.

In many cases, anomalous events occur through valid traffic. A compromised account can generate traffic that follows protocol rules, while the sequence of actions differs from normal usage. In this context, traffic-based models may fail to detect anomalies.

The problem addressed in this thesis is the lack of integration between user behavior and network traffic in anomaly detection. The challenge is to represent both sources of information in a common space and use this representation to detect deviations.

1.3. Scope and Limitations

This thesis studies anomaly detection in network traffic using behavioral profiling and deep learning. The analysis focuses on the integration of behavioral data and traffic data through representation learning.

The behavioral dataset is obtained from monitored web activity. It represents interaction patterns but does not include all possible forms of user behavior. The traffic datasets represent network flows in controlled environments.

The study does not include real-time deployment. It does not analyze encrypted payload content. The association between behavior and traffic is based on similarity in latent space and does not represent direct correspondence between users and traffic records.

1.4. General Objective

To develop a method for anomaly detection in network traffic through the integration of user behavioral profiling and unsupervised representation learning.

1.4.1 Specific Objectives

1. To collect web activity data and represent user interaction through feature vectors and latent representations.
2. To align behavioral embeddings and network traffic features in a shared representation space.
3. To design and train deep learning models that use enriched representations for anomaly detection.
4. To compare models using traffic data and models using enriched data under common evaluation metrics.

1.5. Methodology

This thesis follows an experimental methodology based on data collection, model development, and evaluation. The study uses two types of data. The first type is behavioral data obtained from monitored web activity. The second type is network traffic data obtained from benchmark datasets. The population of the study consists of 150 records divided into four groups: network analysts, programmers, mechatronics students, and a fourth group of records labeled as atypical (anomalous) user behavior. Each record corresponds to a monitored or generated web activity session.

The methodology includes several stages. First, behavioral data is collected and transformed into feature vectors. Second, latent representations are learned using autoencoders. Third, clustering methods are applied to construct behavioral profiles. After this stage, behavioral embeddings are integrated with traffic data through an alignment process in latent space. Finally, deep learning models are trained and evaluated using the enriched dataset. The evaluation uses classification metrics such as accuracy, precision, recall, and F1 score.

Before any experiment runs, the pipeline separates the dataset into training, validation, and test partitions, and applies scaling, class balancing, autoencoder training, and adversarial alignment only to the training partition. An automatic check follows every data-loading step and halts execution if a label-like column, such as label, binary_label, attack_cat, id, or cluster, appears

among the input features. Chapter 5 describes these two safeguards in full and traces them to a leakage problem found in an earlier version of the pipeline.

1.5.1 Technological Tools

The study uses software tools for data collection, processing, and model training. Behavioral data is collected using Charles Proxy, which captures HTTP and HTTPS traffic. Data processing and model development are performed using programming environments for machine learning. Neural models are implemented using deep learning frameworks. Data analysis and preprocessing are carried out using numerical computing libraries.

These tools support the construction of datasets, the training of models, and the evaluation of results.

All experiments were conducted on Google Colab using an NVIDIA Tesla T4 GPU. The source code developed for this study is available as a supplementary resource in Appendix B.

1.6. Thesis Organization

This doctoral thesis is organized into seven chapters.

Chapter 1 constitutes the present introduction. It describes the motivation of the work, the problem statement, the scope and limitations, the general and specific objectives, the methodology employed, and the technological tools used.

Chapter 2, Theoretical Foundations, presents the fundamental concepts required for the development of the work. It covers the foundations of networks and data, network communication models and protocols, the representation of traffic as observational data, user behavior in network environments, and data representation in feature spaces. It also addresses machine learning foundations, including learning paradigms, representation learning and latent spaces, classification functions, and similarity measures.

Chapter 3, User Behavioral Profiling for Network Security, introduces the study of user behavior in network environments. It explains the idea of behavioral profiling and reviews research

on user behavior and anomaly detection. The chapter also describes the data collection process based on monitored web activity. Next, it presents the construction of behavioral features and the generation of behavioral embeddings through representation learning. The chapter then describes the clustering methods used to identify user profiles and patterns of network use.

Chapter 4, Cross-Domain Behavioral Enrichment for Network Traffic, presents the method used to combine user behavior and network traffic. It begins with a brief background on anomaly detection in network traffic and reviews related work on machine learning methods for intrusion detection. The chapter then describes the datasets used in the study. After this, it explains the enrichment process. This includes the integration of behavioral embeddings, the alignment of representations through adversarial learning, and the creation of enriched datasets that link traffic features with behavioral information.

Chapter 5, Deep Learning Models for Network Anomaly Detection, describes the model used to detect anomalies. It explains how enriched features are represented and how they are used as input for the model. The chapter then presents the architecture of the deep feedforward neural network and the training process used to learn patterns of network activity. It also describes the evaluation framework and the metrics used to measure model performance. The chapter also compares models trained with traffic features only and models trained with enriched data that include behavioral information. In addition, it examines the results obtained with different network traffic datasets.

Chapter 6, Discussion, examines the results of the whole set of experiments. It discusses what the results show about the use of behavioral information in network anomaly detection. The chapter also considers the limits of the proposed approach and the role of behavioral context in the analysis of network activity.

Chapter 7, Conclusions and Future Work, summarizes the main results of the thesis. It reviews the contributions of the work and outlines directions for future research related to user behavior analysis, representation learning, and anomaly detection in network environments.

2. Theoretical Foundations

2.1. Network and Data Foundations

This section presents the fundamental concepts related to computer networks and the generation of network traffic. It defines communication models, protocols, and traffic flows as the basis for data generation in network environments. It also introduces the concept of network traffic as a source of observations and describes how user interaction with services produces measurable data. The section establishes the relation between network activity and data representation without focusing on specific datasets or implementations.

2.1.1. Network Communication Models and Protocols

Computer networks are systems that enable the exchange of data between interconnected devices. The design of these systems is governed by layered communication models, which organize the functions of a network into distinct levels of abstraction. The most widely adopted reference model is the Open Systems Interconnection (OSI) model, which decomposes network communication into seven layers: physical, data link, network, transport, session, presentation, and application [ISO-84]. Each layer is responsible for a specific set of functions and communicates with adjacent layers through defined interfaces. This architecture allows different hardware and software implementations to interoperate, provided they conform to the specifications of each layer.

The Internet, as a practical instantiation of layered communication, follows the TCP/IP protocol suite, which consolidates the OSI layers into four functional groups: network access, internet, transport, and application. The Internet Protocol (IP) operates at the network layer and is responsible for the addressing and routing of packets across heterogeneous networks [Cerf-Kahn-74]. IP provides a best-effort, connectionless delivery service: packets are forwarded independently and may arrive out of order or be lost in transit. The Transmission Control Protocol (TCP) operates above IP and provides a reliable, connection-oriented byte stream by managing sequencing, acknowledgment, and retransmission [Cerf-Kahn-74]. The User Datagram Protocol (UDP) offers an alternative transport service without connection establishment or reliability guarantees, suited to applications where low latency is preferred over guaranteed delivery.

Application-layer protocols define the formats and procedures for specific types of communication. The Hypertext Transfer Protocol (HTTP) governs the exchange of documents and resources on the World Wide Web, following a request-response model between clients and servers. The Domain Name System (DNS) translates human-readable hostnames into network addresses, enabling the resolution of service identifiers to routable destinations. These protocols produce structured interaction patterns in the network, as they impose defined sequences of messages and connection behaviors that are observable in traffic records.

The layered structure of network protocols has direct implications for traffic analysis. Each layer adds headers and metadata to transmitted data units, producing a hierarchical record of the communication. This record encodes information about endpoints, timing, protocol state, and application intent. Traffic observed at a network monitoring point contains fields derived from multiple protocol layers simultaneously, and the analysis of these fields allows the characterization of network flows. A flow, in this context, is defined as a sequence of packets sharing common attributes such as source and destination addresses, port numbers, and transport protocol, within a bounded time interval [Cerf-Kahn-74]. The flow abstraction aggregates packet-level detail into a compact representation suitable for analysis.

2.1.2. Network Traffic as Observational Data

Network traffic is the aggregate of all data transmitted across a network within a period of time. From an analytical perspective, traffic constitutes a sequence of observations generated by the interactions of users, applications, and services. Each observation encodes properties of a communication event: the identities of the communicating endpoints, the volume of data exchanged, the timing of the exchange, and the protocol used. These properties are measurable and can be extracted from packet captures or summarized into flow records.

The statistical structure of network traffic reflects the behavior of the underlying processes that generate it. Traffic arising from human activity tends to exhibit temporal patterns related to usage habits, such as concentration during working hours and reduced volume at night. Within a session, the distribution of packet sizes and inter-arrival times depends on the application generating the traffic. Web browsing produces bursty exchanges of relatively small requests followed by larger responses. File transfers generate sustained flows with large payloads.

Streaming applications produce flows with periodic packet arrivals and variable bitrates. These characteristic patterns make it possible to associate observed traffic with the type of application or behavior that generated it [Trevisan-Giordano-20].

Network traffic is also a high-dimensional data source. A single flow record may include dozens of features derived from packet headers and timing measurements. These features include byte counts, packet counts, duration, mean and variance of packet sizes, inter-arrival times, flag combinations, and protocol identifiers. The joint distribution of these features constitutes the observational space from which patterns are extracted. The high dimensionality of this space implies that many flows may be similar in some dimensions while differing in others, and that the identification of meaningful structure requires methods capable of handling such complexity.

An important property of network traffic as observational data is its non-stationarity. The statistical properties of traffic change over time as user populations evolve, new applications are deployed, and network infrastructure changes. Traffic distributions observed during one period may not accurately represent those of a later period. This non-stationarity creates challenges for models trained on historical data, as they may encounter conditions not present in the training distribution. The characterization of traffic therefore requires attention to temporal context and the possibility that learned patterns may degrade in relevance over time [Lakhina-04].

2.1.3. User Behavior in Network Environments

Users generate network traffic through their interaction with services and applications. Each interaction produces a record that can be observed at the network level. A user browsing a website issues HTTP requests to a server, and the resulting exchanges are recorded as network flows. A user connecting to a messaging application initiates and maintains sessions that produce periodic traffic. These observable behaviors depend on the choices, habits, and schedules of the user, and they constitute a behavioral signal embedded within the traffic.

The concept of user behavior in network environments refers to the patterns of interaction between users and network services, as observable through traffic records or application logs. Users tend to interact with a consistent set of services, at predictable times, and with characteristic frequency distributions across different resources. A user who regularly accesses a specific set of web domains, at similar times each day, and with similar volumes of data, produces a behavioral

pattern that can be described statistically. This pattern represents the habitual activity of the user and serves as a reference against which future observations can be compared.

Behavioral patterns are shaped by the role and context of the user within the network. In organizational environments, users tend to interact with services relevant to their professional tasks: internal applications, communication platforms, and domain-specific resources. Deviations from these patterns may arise from changes in job function, from the adoption of new tools, or from the actions of an adversary who has obtained access to the user's credentials. The study of user behavior in network environments thus provides a lens through which changes in the nature of network activity can be interpreted [Denning-87].

The aggregation of behavioral observations over time allows the construction of a behavioral profile for each user. A behavioral profile is a statistical summary of the interactions of a user with network services, derived from observed activity over a reference period. Profiles can be represented as vectors of aggregated features, as probability distributions over service categories, or as sequences of events. The form of the representation determines which aspects of behavior are captured and which are abstracted away. Regardless of the representation chosen, the profile encodes the characteristic structure of normal activity for a given user and provides a basis for identifying deviations.

2.1.4. Data Representation in Feature Spaces

The transformation of raw network observations into structured data suitable for analysis requires the definition of a feature space. A feature space is a mathematical space in which each observation is represented as a vector of numerical values, each corresponding to a measurable attribute of the observation. The choice of features determines what information is captured and what is discarded. Well-chosen features preserve the information relevant to the analytical task while reducing irrelevant variability.

In network traffic analysis, features are derived from packet-level measurements and flow-level statistics. Packet-level features include header fields such as source and destination ports, protocol identifiers, and TCP flags. Flow-level features aggregate packet measurements into statistics: total byte count, packet count, mean and variance of inter-arrival times, duration, and ratios between forward and backward traffic directions. These features transform a variable-length

sequence of packets into a fixed-length vector, enabling the application of standard analytical methods.

The choice of representation involves trade-offs between expressiveness and tractability. A representation with many features can capture fine distinctions between observations but may introduce noise and increase the cost of computation. A representation with few features may fail to distinguish between observations that differ in important ways. Dimensionality reduction methods address this trade-off by projecting high-dimensional feature vectors into lower-dimensional spaces that retain the most informative structure [Greenacre-Groenen-22]. Principal component analysis, for example, finds a linear projection that maximizes the variance of the projected data. Other methods seek non-linear projections that preserve local structure or maximize statistical independence.

Feature spaces also define the geometry within which similarity between observations is measured. Two observations that are close in feature space share similar values across all measured attributes, and are therefore considered similar in behavior. Distance functions quantify this proximity and provide the basis for clustering, classification, and anomaly detection methods. The Euclidean distance, the cosine similarity, and the Mahalanobis distance are among the most common measures used in traffic analysis. The appropriateness of a given distance measure depends on the distribution of the data and the structure of the feature space.

When observations originate from different sources or measurement contexts, their feature spaces may differ in distribution even when the underlying processes are similar. Aligning representations from different domains allows observations from one context to be compared with those from another. This alignment is relevant when behavioral data and traffic data, collected through different measurement processes, must be analyzed jointly within a common representational framework. The construction of a shared feature space that integrates heterogeneous observations is a central challenge in multi-source data analysis.

2.2. Machine Learning Foundations

This section presents the theoretical basis of machine learning for pattern analysis. It defines

learning as a process to model data through mathematical functions and optimization. It introduces core concepts such as feature space, latent representations, classification, clustering, and similarity. The section explains how models identify structure in data and how decision functions are used to separate classes. It provides the theoretical framework required to understand anomaly detection without describing specific methods or architectures.

2.2.1. Learning Paradigms

Machine learning is the study of algorithms that improve their performance on a task through experience [Mitchell-99]. In the context of data analysis, the task is to identify structure, make predictions, or generate representations from observed data. The algorithm learns by adjusting its internal parameters to minimize a criterion that measures discrepancy between its output and a desired behavior. The parameters are adjusted using optimization procedures applied to a dataset of observed examples.

The principal paradigms of machine learning differ in the type of supervision available during learning. In supervised learning, each training example is associated with a label that specifies the desired output. The algorithm learns a mapping from input features to outputs by minimizing the discrepancy between predicted and true labels across the training set [Vapnik-95]. Supervised learning requires labeled data, which in network security contexts may be difficult to obtain in quantity, since the annotation of attack traffic requires expert knowledge and the collection of representative attack examples.

Unsupervised learning does not use labels. The algorithm identifies structure in the data without guidance from external annotations. Clustering methods partition the data into groups of similar observations. Density estimation methods model the probability distribution of the data. Dimensionality reduction methods find compact representations that preserve the informative structure of the data. These methods are applicable when labeled data is unavailable or when the goal is to characterize the distribution of observations rather than to assign predefined categories.

Semi-supervised learning combines a small amount of labeled data with a large amount of unlabeled data. The labeled examples provide information about the mapping from inputs to outputs, while the unlabeled examples constrain the structure of the learned representation. This paradigm is relevant in settings where labeling is costly and most available data is unannotated.

Self-supervised learning is a related approach in which the learning signal is derived from the data itself, without manual annotation, by defining pretext tasks whose solutions require the learning of useful representations.

2.2.2. Representation Learning and Latent Spaces

A representation is a transformation of raw data into a form that is suitable for analysis. The quality of a representation determines how easily subsequent methods can identify relevant structure in the data. A representation that places similar observations close together in a vector space and separates dissimilar observations facilitates both classification and clustering. The learning of representations from data, rather than their manual design, is the subject of representation learning.

Representation learning methods train a model to transform input observations into vectors in a lower-dimensional latent space. The latent space is a continuous vector space in which the position of a point encodes the properties of the corresponding observation. If the model is trained to reconstruct its inputs from their latent representations, the latent vectors must capture the information necessary for reconstruction. If the model is trained to discriminate between categories, the latent vectors must encode the features relevant to discrimination. The structure of the latent space depends on the objective used during training [Bengio-13].

Autoencoders are a class of models that learn representations by training an encoder to map inputs to latent vectors and a decoder to reconstruct inputs from latent vectors. The encoder and decoder are trained jointly to minimize reconstruction error. The bottleneck structure of the autoencoder, in which the latent space has lower dimensionality than the input space, forces the model to learn a compact representation that captures the most important structure of the data [Goodfellow-Bengio-Courville-16]. The resulting latent vectors can then be used as features for downstream analysis tasks.

The geometry of the latent space reflects the learned structure of the data. Observations that are close in latent space were mapped there because the model found them to share relevant properties. Distances in the latent space can therefore serve as measures of similarity that are more informative than distances in the original feature space, particularly when the original space contains irrelevant or redundant dimensions. The alignment of latent spaces from different data sources allows the comparison of observations collected under different conditions, enabling the

joint analysis of heterogeneous data.

2.2.3. Classification and Decision Functions

Classification is the task of assigning an observation to one of a predefined set of categories. A classifier learns a decision function that maps feature vectors to category labels. The decision function partitions the feature space into regions, each associated with one category. When a new observation is presented to the classifier, its feature vector is mapped to one of these regions, and the corresponding label is assigned.

The decision function can take many forms depending on the family of classifiers used. Linear classifiers define decision boundaries as hyperplanes in the feature space [Vapnik-95]. Non-linear classifiers use curved boundaries that can separate categories whose distributions overlap in complex ways. The capacity of a classifier, its ability to represent complex decision functions, must be matched to the complexity of the problem: insufficient capacity leads to underfitting, in which the classifier fails to capture relevant structure, while excessive capacity leads to overfitting, in which the classifier captures noise specific to the training data.

The performance of a classifier is measured by comparing its predictions with true labels on held-out data. Standard evaluation criteria include accuracy, precision, recall, and the F1 score. Precision measures the proportion of positive predictions that are correct. Recall measures the proportion of true positives that are correctly identified. The F1 score is the harmonic mean of precision and recall, providing a single scalar summary of classifier performance when both false positives and false negatives are relevant costs. In anomaly detection, the class imbalance between normal and anomalous observations affects the interpretation of these metrics, and weighted or threshold-adjusted variants are often used.

Training a classifier requires choosing a loss function that quantifies the cost of incorrect predictions and an optimization algorithm that adjusts model parameters to minimize this loss. The cross-entropy loss is widely used for classification problems, as it penalizes confident incorrect predictions more heavily than uncertain ones. Optimization is performed using gradient-based methods that iteratively adjust parameters in the direction that reduces the loss.

2.2.4. Similarity Measures and Distance Functions

The measurement of similarity between observations is fundamental to many machine learning methods. Clustering groups observations by similarity. Nearest-neighbor classifiers assign labels based on the categories of the most similar training examples. Anomaly detection identifies observations that are dissimilar from a reference set of normal examples. The choice of similarity measure determines which observations are considered alike and which are considered different, and therefore shapes the structure identified by these methods.

The Euclidean distance between two vectors measures the straight-line distance in the feature space. It is sensitive to differences in all dimensions equally and is appropriate when features are measured on comparable scales and when the relevant structure of the data is well captured by linear geometry. The Mahalanobis distance extends the Euclidean distance by accounting for the covariance structure of the data, normalizing each dimension by its variance and removing correlations between dimensions. This measure is more appropriate when features have different scales or are correlated.

The cosine similarity between two vectors measures the cosine of the angle between them, ignoring their magnitudes. Two vectors are considered similar if they point in the same direction in the feature space, regardless of their lengths. Cosine similarity is widely used in text analysis and in settings where the direction of a feature vector is more informative than its magnitude. In network traffic analysis, cosine similarity is applicable when behavioral profiles are represented as frequency vectors, since the relative distribution of activity across services is more informative than its absolute volume [Aggarwal-16].

The choice of distance function also determines the geometry used by clustering algorithms. The k-means algorithm [MacQueen-67] minimizes the sum of squared Euclidean distances between observations and their assigned cluster centroids. Clustering methods based on graph connectivity or density, such as DBSCAN [Ester-96], define similarity in terms of local density and reachability rather than global distance, making them less sensitive to the global scale of the data. The selection of an appropriate distance function is therefore an integral part of the design of any analysis pipeline that relies on the comparison of observations.

Information-theoretic measures provide an alternative basis for assessing similarity between distributions. The Kullback-Leibler divergence quantifies the difference between two

probability distributions and is used in methods that model observations as samples from underlying distributions [Cover-Thomas-06]. The Jensen-Shannon divergence, a symmetrized variant, provides a bounded measure of distributional similarity. These measures are applicable when behavioral profiles are represented as probability distributions over service categories, enabling the comparison of users with similar distributional patterns of activity.

3. User Behavioral Profiling for Network Security

3.1. Introduction

Computer networks support communication, data exchange, and access to digital services. These systems process large volumes of traffic generated by users and applications. The increase in network activity creates new challenges for monitoring and security [Trevisan-Giordano-20; Wang-Balarezo-21]. Attacks, misuse of resources, and unauthorized access may appear within normal traffic flows. For this reason, the detection of anomalous activity has become a central task in cybersecurity [Wang-Balarezo-21].

Many detection systems analyze traffic patterns to identify deviations from normal operation [Li-Rios-21; Wang-Balarezo-21; Hooshmand-Hosahalli-22]. These approaches examine packet flows, connection statistics, and protocol behavior. While traffic analysis provides useful information, it does not always capture how users interact with network services. Attacks may occur through legitimate services when user accounts are compromised or misused. In such cases, the analysis of user behavior provides additional context for identifying unusual activity [Martin-Fernandez-21].

User behavioral profiling addresses this limitation by modeling patterns of interaction between users and network services [Salem-Stolfo-11; Martin-Fernandez-21; Tao-Lu-24]. A behavioral profile represents how a user typically accesses resources such as websites, applications, and network services. These profiles allow the system to define a reference pattern of normal activity. When behavior deviates from this reference, the deviation may indicate misuse, intrusion, or abnormal activity. For this reason, behavioral profiling supports the detection of anomalies that cannot be observed through traffic features alone.

The study of behavioral patterns has become relevant as Internet usage grows and user activity becomes more diverse [Trevisan-Giordano-20; Rani-Gill-23]. Users interact with multiple platforms such as educational systems, development tools, and communication services. Each user produces a sequence of actions that reflects habits of interaction with the network. Modeling these patterns allows the identification of differences between users and supports the detection of abnormal activity within the system.

This chapter presents the study of user behavior in network environments. It explains how user activity can be described through behavioral profiles. Network use often follows patterns that reflect how users interact with services and applications. When activity moves away from these patterns, the change may indicate unusual behavior. For this reason, user behavior can support the analysis of network activity.

This chapter also presents the process used to represent and analyze user behavior. It describes the collection of network activity from monitored users and the construction of behavioral features from this data. These features are used to build behavioral vectors, learn latent representations, and group users through clustering methods. The resulting profiles describe patterns of network use and provide the basis for the analysis developed in the following chapters.

3.2. Background on User Behavior in Networks

Network security requires the analysis of user activity. Users interact with services, applications, and web platforms. These interactions produce network traffic. Traffic records show how users access resources in a network. From these records, it is possible to observe patterns of use. The analysis of these patterns helps identify abnormal behavior.

User behavior in networks can be studied through behavioral profiling. Behavioral profiling builds representations of how users interact with services over time. These representations are obtained from observable network activity. Examples include HTTP requests, application access, connection frequency, and session records. The aggregation of these signals produces a behavioral description for each user. This description summarizes how each user interacts with network services. Behavioral profiling is therefore used in network monitoring and anomaly detection systems [Salem-Stolfo-11].

Behavioral profiling assumes that normal activity follows patterns. When user actions change from these patterns, the change may indicate abnormal behavior. Such changes can occur when an attacker uses a compromised account or when a user accesses services in a different way from usual activity. In these cases, traffic may appear valid at the protocol level. However, the sequence of actions may differ from the expected behavior of the user. Behavioral analysis allows

the detection of these differences [Martin-Fernandez-21; Tao-Lu-24].

User behavior can be described through measurable features. These features include the number of accesses to services, the distribution of requests across applications, and the duration of user sessions. These measures allow the construction of behavioral vectors. Each vector represents the activity of a user across services in the network. Each dimension of the vector corresponds to one observable interaction with a service or application. This representation allows the use of machine learning methods to analyze user behavior.

Many behavioral analysis approaches use unsupervised learning. In network security, labeled data about malicious behavior is limited. For this reason, clustering methods are often used to group users with similar patterns of activity. Each group represents a behavioral profile. These profiles define reference patterns of network use. New activity can then be compared with these patterns in order to detect deviations or anomalies. Similar approaches have been used in insider threat detection and masquerade detection studies [Salem-Stolfo-11; Singh-Jindal-23].

Behavior analysis also requires a numerical representation of user activity. Network logs contain records of user interactions with services. From these records, feature vectors can be constructed. Each feature captures one aspect of user interaction. These vectors allow the application of machine learning models to study patterns in user behavior.

Some studies apply representation learning to transform behavioral vectors into latent representations. Dimensionality reduction methods can project behavioral data into a smaller feature space. In this space, patterns of user activity can be analyzed with clustering or classification methods. These representations help capture relationships among user actions.

Deep learning methods have also been used to learn behavioral representations from network activity. Models such as autoencoders can learn latent structures from interaction data. These structures summarize patterns of activity present in the behavioral dataset. These representations can then be used as input for anomaly detection systems [Tao-Lu-24].

Behavioral profiling also provides information about how users interact with network services. Users access many platforms during normal activity. These may include educational systems, development tools, collaborative platforms, and communication services. Each user

produces a sequence of interactions with these services. The analysis of these sequences allows the identification of patterns of network use.

Behavior analysis complements traditional traffic analysis. Traffic analysis focuses on packet features and protocol behavior. Behavioral analysis focuses on user interaction with services. The combination of both perspectives supports the detection of abnormal activity in network environments.

In this thesis, user behavior is analyzed through interaction patterns extracted from monitored web activity. These records describe how users access network services during an observation period. From these records, behavioral representations are constructed for each user. These representations are used to identify patterns of network use. The resulting behavioral profiles provide the basis for the representation learning and clustering processes described in the following sections.

3.3. Behavioral Data Collection

The behavioral dataset used in this study was obtained through direct monitoring of user web activity. Network traffic generated during user interaction with online services was captured and logged for later analysis. The goal of this collection process was to obtain observable interaction patterns that reflect how users access different web platforms during normal usage sessions.

Traffic collection was performed using Charles Proxy, a debugging proxy tool designed to capture and inspect HTTP and HTTPS communication between client applications and remote servers. Charles Proxy acts as an intermediary between the user device and the network. All requests and responses pass through the proxy, allowing the system to record detailed information about each interaction without modifying the network infrastructure.

The software used in this study was Charles Proxy version 5.0.3, developed by XK72 Ltd. (Wellington, New Zealand). The tool is publicly available at: https://www.charlesproxy.com

Charles Proxy enables the inspection of application-layer communication by intercepting web traffic generated by browsers and software applications. Because the proxy operates at the

HTTP/HTTPS layer, it can capture requests produced by web browsing sessions and application interactions in real time. This approach allows traffic monitoring without interrupting the user session or modifying network routing configurations.

During data collection, Charles Proxy recorded multiple attributes associated with each request. These attributes include:

- destination domain or URL
- HTTP request headers
- cookies and session identifiers
- request timestamps
- response metadata
- connection information

These logs provide a structured record of user interaction with network services. Each request captured by the proxy corresponds to an interaction between a user application and a remote service. When aggregated over time, these records describe how users distribute their activity across different platforms.

The proxy-based monitoring approach was selected because it allows traffic capture at the application level while preserving normal user activity. Alternative tools such as packet analyzers capture low-level packet data but often require network-level configuration or produce data that is more difficult to associate with specific user actions. Proxy monitoring provides a direct mapping between HTTP requests and the services accessed by users, which simplifies behavioral analysis.

The behavioral dataset was collected from 150 participants. Each participant used a monitored workstation during an observation session of approximately 30 minutes. During this time, users interacted with different web platforms as part of normal activities such as browsing documentation, accessing educational platforms, using development tools, and interacting with online resources.

The total observation time across all participants was approximately 4500 minutes, equivalent to 75 hours of network activity. Participants were grouped according to their academic or professional background in order to capture variation in network usage patterns. The study

included four user groups:

1. 20 network analysts, whose activities involved network management tools and technical resources
2. 10 programmers, whose sessions included development environments and programming platforms
3. 60 mechatronics students, who interacted with educational platforms and technical resources related to engineering studies
4. 60 records labeled as atypical (anomalous) user behavior, included to provide examples of deviating usage patterns for the profiling and clustering analysis

This distribution allowed the dataset to capture different patterns of interaction with online services. Each group generated traffic associated with different types of platforms, which introduced diversity in the collected behavioral data.

All participants provided consent to participate in the study before data collection began. The monitoring process recorded only interaction metadata and did not collect personal content or credentials. The objective was to observe access patterns to web services rather than to analyze user-specific private information.

The collected traffic logs were then organized according to the services accessed during the sessions. Each monitored service corresponds to a domain or platform that appeared in the captured traffic. Table 3.1 lists the services that were accessed by participants during the monitoring period.

Each request recorded by Charles Proxy was associated with one of these domains. Requests were later aggregated to represent the interaction frequency of each user with each monitored service. This aggregation allowed the construction of behavioral descriptions that summarize how users distribute their activity across different platforms.

The resulting dataset consists of structured records derived from HTTP and HTTPS traffic captured during the monitoring sessions. These records provide the foundation for the behavioral modeling process described in the following sections.

TABLE 3.1

SERVICES ACCESSED DURING BEHAVIORAL DATA COLLECTION

Service	Description
Teams	Microsoft Teams collaboration and communication platform
Thingiverse	Platform for sharing 3D design models
Multisim	Electronic circuit simulation software
Canvas	Learning management system
Wikipedia	Online collaborative encyclopedia
NetAcad	Cisco networking education platform
Arduino	Development environment for embedded systems
MATLAB	Numerical computing environment
Stack Overflow	Programming question-and-answer platform
SevOne	Network monitoring platform
Packet Tracer	Cisco network simulation software
Spectrum	Network performance monitoring platform
ServiceNow	IT service management platform
Visual Studio	Software development environment
Udemy	Online learning platform
Xcode	Development environment for Apple platforms
GitHub	Version control and collaborative development platform

The behavioral dataset therefore reflects real interaction patterns generated during normal web activity. These patterns serve as the basis for constructing behavioral representations of users and for studying how interaction habits relate to network activity patterns. The resulting dataset contains 150 records, one per participant. Each record includes a user identifier, an IP address, raw interaction counts for each of the 17 monitored services, and a category label indicating the user group (1 = mechatronics student, 2 = programmer, 3 = network analyst, 4 = atypical/anomalous-

behavior record). Table 3.2 shows a representative excerpt of the dataset.

TABLE 3.2

EXCERPT FROM THE BEHAVIORAL DATA COLLECTION DATASET

User	Teams	Thing.	Mult.	Can.	Wiki	NetAcad	Ardu.	MATL.	Stack	Sev1	PTra.	Spect.	SNow	VStud.	Udemy	Xcode	GitHub	Cat.
E1	4	3	2	5	2	1	2	1	2	0	2	0	0	3	0	1	0	1
E2	3	4	3	4	1	1	2	1	1	0	3	0	0	2	0	0	0	1
E3	3	3	1	3	1	2	0	0	1	0	1	0	0	4	0	0	0	1
E4	1	2	1	5	0	1	1	2	1	0	1	0	0	2	0	0	2	1
E5	2	2	1	3	0	1	0	1	1	0	1	0	0	2	0	0	0	1

Each row corresponds to one participant. Each numeric value represents the raw count of interactions recorded with that service during the monitoring session. The dataset comprises 150 samples in total: 60 mechatronics students, 10 programmers, 20 network analysts, and 60 records labeled as atypical (anomalous) user behavior. These raw counts are subsequently normalized to produce relative interaction frequencies, as described in Section 3.4.1.

3.4. Behavioral Profile Construction

This section describes the process used to construct behavioral profiles from the collected web activity data. The process consists of four stages: feature construction, representation learning, clustering of behavioral embeddings, and construction of the behavioral profile dataset. Each stage transforms the behavioral data into a more structured representation that supports the analysis of user activity patterns.

3.4.1. Behavioral Feature Construction

The behavioral dataset collected from proxy logs contains records of user interactions with different web services. Each record corresponds to a request generated during a user session. These records include the accessed domain, the time of the request, and other connection metadata captured by the proxy system.

Raw interaction logs cannot be used directly by machine learning models. The records must first be transformed into numerical features that represent user activity in a structured form. This process converts the set of HTTP requests generated during each session into a behavioral representation that describes how a user interacts with different services.

The feature construction process begins by counting the number of interactions with each monitored service. For each user u_i, the total number of requests to each service j is recorded. Let x_{ij} denote the number of interactions between user i and service j. The complete set of interactions for each user forms the initial activity vector.

Users generate different numbers of requests during their sessions. Some users access many services, while others interact with a smaller set of platforms. To allow comparison between users,

the interaction counts are normalized. Normalization converts raw counts into relative frequencies so that each service reflects its proportion of the total user activity.

The normalized frequency of interaction with service j for user i is defined as

$$f_{ij} = \frac{x_{ij}}{\sum_{j=1}^{n} x_{ij}}.$$

where f_{ij} represents the normalized frequency of service j for user i, x_{ij} is the number of interactions with that service, and

$$\sum_{j=1}^{n} x_{ij},$$

corresponds to the total number of interactions generated by the user. This normalization step allows the activity of users with different interaction volumes to be represented on a comparable scale [Salem-Stolfo-11].

After normalization, the distribution of activity across services is analyzed through an entropy measure. Entropy quantifies the variability of user interaction patterns. A user who accesses many services in similar proportions produces a higher entropy value, while a user who focuses on a small set of services produces a lower entropy value.

The entropy value for each user is calculated using Shannon's entropy formulation:

$$H_i = -\sum_{j=1}^{n} p_{ij} \log(p_{ij}).$$

where H_i represents the entropy of user i, p_{ij} is the probability of interaction with service j, and n corresponds to the number of monitored services. The probability p_{ij} is obtained from the normalized interaction frequencies:

$$p_{ij} = \frac{x_{ij}}{\sum_{j=1}^{n} x_{ij}}.$$

Entropy measures the dispersion of user activity across services. This value captures the degree to

which a user concentrates activity on a small number of platforms or distributes activity across several services [Shannon-48].

After calculating interaction frequencies and entropy values, dimensionality reduction is applied to summarize the structure of the behavioral dataset. The objective of this step is to extract a compact representation that preserves the main variance in the data while reducing the number of dimensions used in later stages of the analysis.

Principal Component Analysis (PCA) is used to obtain two principal components from the behavioral feature space. PCA transforms the original variables into a set of orthogonal components that capture the main directions of variance in the dataset [Greenacre-Groenen-22].

Let X denote the matrix of normalized behavioral features, where each row corresponds to a user and each column corresponds to a service interaction frequency. PCA computes a projection of X into a lower-dimensional space defined by the eigenvectors of the covariance matrix of X. The projection can be expressed as

$$Z = XW,$$

where Z represents the transformed data matrix and W contains the eigenvectors associated with the principal components.

In this study, the first two principal components are extracted. These components capture the largest sources of variation in the behavioral dataset. Instead of replacing the original variables, the PCA components are appended to the dataset. This approach preserves the detailed service interaction features while adding a compact representation of the overall activity structure. The final behavioral feature set therefore contains three types of information for each user:

1. normalized interaction frequencies for each monitored service
2. the entropy value describing the distribution of activity
3. the first two principal components derived from PCA

Together, these features provide a structured description of user interaction patterns. The resulting dataset serves as the input for the behavioral representation learning process described in the next section.

The behavioral features constructed in this section provide a structured description of how users interact with network services. Interaction frequencies capture the distribution of requests across platforms. The entropy measure captures how user activity spreads across services. The principal components capture patterns that appear in the global structure of the dataset. Together, these variables describe the activity profile of each user in numerical form. This representation allows the comparison of user activity patterns within the dataset.

This feature construction stage prepares the behavioral dataset for representation learning. The resulting feature matrix contains the interaction structure of each user session and allows the application of learning methods that identify patterns in the data. The next section introduces the representation learning process used in this study. In that section, behavioral features are used to learn latent representations that summarize user activity patterns through neural models.

3.4.2. Behavioral Representation Learning

The behavioral features constructed in the previous section describe how each user interacts with network services. These features include interaction frequencies, entropy values, and principal components obtained from the behavioral dataset. Together, these variables represent user activity patterns in numerical form. Given that the feature space may contain redundancy and correlations between variables, a representation learning step is used to obtain a compact representation of user behavior.

Representation learning aims to transform the original feature space into a latent space that preserves the main structure of the data. Neural models can learn this transformation directly from the input data. In this study, an autoencoder model is used to learn latent representations of the behavioral dataset.

An autoencoder is a neural network that learns to reconstruct its input through a compressed internal representation. The model consists of two main components: an encoder and a decoder. The encoder maps the input vector into a latent representation, while the decoder reconstructs the input from this latent vector. Let $x_i \in \mathbb{R}^n$ denote the behavioral feature vector of user i. The encoder function transforms this vector into a latent representation z_i:

$$z_i = f_\theta(x_i).$$

where f_θ represents the encoder function and θ denotes the parameters of the neural network. The decoder then reconstructs the input vector from the latent representation:

$$\hat{x}_i = g_\phi(z_i),$$

where g_ϕ represents the decoder function and ϕ corresponds to the decoder parameters. The objective of the autoencoder is to minimize the difference between the original input x_i and the reconstructed output $\hat{x}_i$. This objective is expressed through the reconstruction loss:

$$L = \|x_i - \hat{x}_i\|^2.$$

This loss function measures the reconstruction error using the mean squared error criterion. During training, the encoder and decoder parameters are adjusted in order to minimize this error. The autoencoder architecture used in this study follows a feedforward structure composed of an encoder and a decoder network. The encoder compresses the behavioral feature vectors into a latent representation, while the decoder reconstructs the original input from this compressed representation. The model is trained using a reconstruction objective based on the mean squared error loss. The main architectural parameters used in the autoencoder are summarized in Table 3.3.

The encoder compresses the behavioral feature vectors into a latent space of lower dimensionality. In this space, each user is represented by a latent vector that summarizes the interaction structure of the original features. This transformation allows the model to capture relationships between behavioral variables that may not be visible in the original feature space. Representation learning using autoencoders has been widely used to extract latent structures in behavioral and network data [Goodfellow-Bengio-Courville-16].

TABLE 3.3

AUTOENCODER ARCHITECTURE USED FOR BEHAVIORAL REPRESENTATION LEARNING

Component	Configuration
Input layer	Behavioral feature vector ($x_i \in \mathbb{R}^n$)
Hidden layer	Dense layer (64 units, ReLU activation)
Latent layer	Dense layer (32 units)
Decoder layer	Dense layer (64 units, ReLU activation)
Output layer	Dense layer (n units)
Loss function	Mean Squared Error
Optimizer	Adam
Epochs	100
Batch size	256

After training the autoencoder, the encoder part of the network is used to obtain the latent representation of each user. These latent vectors form a new dataset where each user is described by a compact behavioral embedding. These embeddings serve as the input for the clustering process described in the next section.

3.4.3. Behavioral Clustering

Once latent behavioral representations are obtained, clustering techniques are used to identify groups of users with similar interaction patterns. Clustering allows the discovery of behavioral structures without requiring labeled data. In this study, clustering is applied to the latent representations generated by the autoencoder.

Let $z_i \in \mathbb{R}^d$ denote the latent representation of user i. The clustering process groups these vectors into a set of clusters:

$$C = \{C_1, C_2, \ldots, C_k\},$$

where each cluster C_j represents a group of users with similar behavioral patterns. Each cluster is represented by a centroid vector μ_j. The centroid corresponds to the mean of the latent vectors belonging to that cluster:

$$\mu_j = \frac{1}{|C_j|} \sum_{z_i \in C_j} z_i,$$

where $|C_j|$ denotes the number of users assigned to cluster j. The assignment of each user vector to a cluster is determined by a distance function. In this study, Euclidean distance is used to measure similarity between behavioral vectors. The cluster assignment for a user i is defined as

$$\hat{C}(z_i) = \arg \min_{j=1,\ldots,k} d(z_i, \mu_j).$$

where $d(\cdot, \cdot)$ denotes the Euclidean distance.

Several clustering algorithms were evaluated in order to identify the most suitable grouping method. These algorithms include K-means clustering [MacQueen-67], agglomerative hierarchical clustering [Tokuda-Comin-22], and density-based clustering using DBSCAN [Ester-96]. The clustering configuration used for behavioral profiling and the main parameters considered during the clustering process are summarized in Table 3.4.

Cluster quality is evaluated using internal validation metrics. The silhouette score measures how well each sample fits within its assigned cluster compared with neighboring clusters [Shahapure-Nicholas-20]. The Davies–Bouldin index measures the similarity between clusters and indicates the level of separation between groups [Davies-Bouldin-79].

Both the silhouette score and the Davies–Bouldin index provide quantitative criteria to determine the clustering configuration that best captures the structure of the behavioral dataset. The clustering process produces groups of users that share similar interaction patterns across the monitored services. Each group represents a behavioral profile that characterizes a specific pattern of network service usage.

TABLE 3.4

CLUSTERING CONFIGURATION USED FOR BEHAVIORAL PROFILING

Parameter	Value
Clustering algorithm	Agglomerative clustering
Distance metric	Euclidean
Linkage method	Ward
Number of clusters	2
Evaluation metrics	Silhouette score, Davies–Bouldin index
Latent input space	Autoencoder embeddings

3.4.4. Behavioral Profile Dataset

The clustering process produces a set of behavioral profiles that summarize the interaction patterns observed in the dataset. Each user is associated with a cluster that represents a behavioral group. These clusters form the basis of the behavioral profile dataset.

Let z_i denote the latent representation of user i, and let $\mu_{C(z_i)}$ denote the centroid of the cluster assigned to that user. The behavioral profile representation is constructed by combining the user embedding with the cluster representation:

$$p_i = [z_i; \mu_{C(z_i)}],$$

where $[\cdot;\cdot]$ denotes vector concatenation. This representation captures both individual behavior and group-level behavior. The latent vector z_i represents the interaction pattern of the user, while the centroid $\mu_{C(z_i)}$ represents the typical pattern of the behavioral group.

The resulting dataset contains one behavioral profile vector for each user in the study. Each profile summarizes the interaction structure observed in the proxy traffic logs and provides a compact representation of user activity patterns.

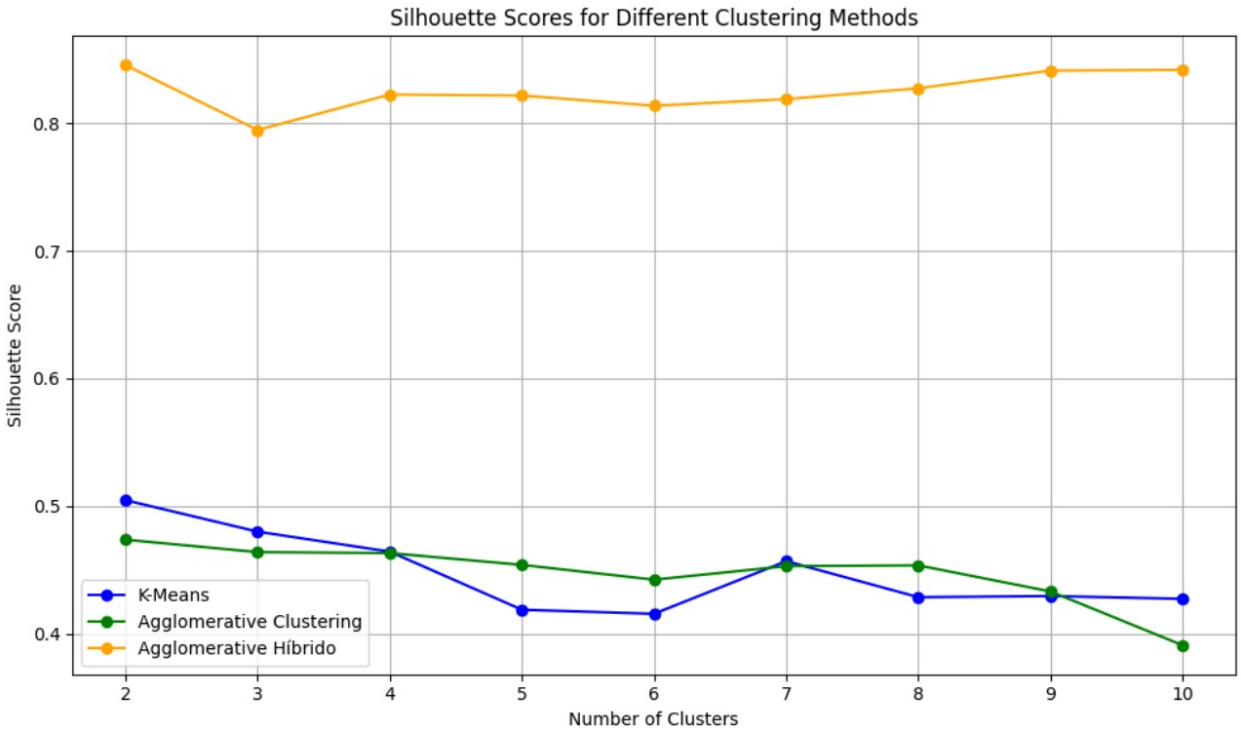


Fig. 3.1 Silhouette score vs number of clusters for K-Means clustering, Agglomerative clustering, and the hybrid model based on an autoencoder and agglomerative clustering. The results show the separation between behavioral groups used for user behavioral profiling.

These behavioral profiles are later used to enrich network traffic datasets through cross-domain alignment. In that process, behavioral embeddings are associated with network traffic records in order to provide contextual information about user activity patterns. This enrichment step allows anomaly detection models to incorporate both traffic characteristics and behavioral context, improving the representation of network activity patterns.

3.5. Behavioral Profiling Results

The behavioral profiling stage transforms raw interaction logs collected with Charles Proxy into structured user profiles. Each record in the proxy log represents a request generated during a user session. These records include domains, timestamps, and connection metadata. The objective of this stage is to group users according to patterns of interaction with network services.

Clustering methods are used to identify groups of users with similar interaction patterns.

These clusters represent behavioral profiles that summarize typical service usage within the dataset. Several clustering algorithms were evaluated to determine which method produces stable behavioral groupings. The evaluation includes K-means clustering, agglomerative clustering, and DBSCAN.

Fig. 3.1 presents the silhouette scores obtained for these clustering methods across different numbers of clusters. The silhouette score measures how well each observation fits within its assigned cluster compared with neighboring clusters. Higher values indicate stronger separation between groups.

As shown in Fig. 3.1, K-means clustering produces silhouette scores between 0.5048 and 0.4159 as the number of clusters increases from two to ten. These values indicate moderate separation between behavioral groups. The clustering structure remains stable across different cluster configurations, although the separation decreases as the number of clusters increases.

Agglomerative clustering shows silhouette scores between 0.4739 and 0.3913. These values follow a pattern similar to the K-means results. The clustering structure remains consistent across cluster numbers but does not produce stronger separation between user groups.

DBSCAN produces silhouette scores between 0.4286 and negative values depending on the parameter configuration. Negative values indicate that some data points are assigned to incorrect clusters. This behavior occurs when the density threshold does not match the distribution of the behavioral data. These results show that DBSCAN does not capture stable behavioral groupings in this dataset.

To improve clustering performance, a hybrid approach was applied. The behavioral features were first processed using an autoencoder to obtain a lower-dimensional latent representation. The clustering algorithms were then applied to this latent space.

The autoencoder compresses the behavioral feature vectors into a reduced representation that preserves the main structure of the data. This latent representation is used as input for clustering. By reducing dimensionality, the clustering algorithm focuses on the most relevant patterns of user activity.

The hybrid approach combines the autoencoder with agglomerative clustering. In this configuration, the autoencoder produces latent vectors that summarize user activity patterns. Agglomerative clustering groups these vectors according to similarity in the latent space.

The results of this hybrid model show a significant improvement in clustering quality. As shown in Fig. 3.1, the silhouette scores obtained with the hybrid model range from 0.7861 to 0.8632. These values are higher than the scores obtained with traditional clustering methods applied directly to the original data.

Higher silhouette scores indicate stronger separation between behavioral groups. The results show that the latent representation generated by the autoencoder captures patterns that are not visible in the original feature space. These patterns allow the clustering algorithm to form more distinct user groups.

Additional clustering metrics support this result. The Calinski–Harabasz index reaches 195.30, indicating separation between groups. The Davies–Bouldin index reaches 0.5775, which reflects limited overlap between clusters. The Dunn index reaches 0.0729, indicating moderate compactness within clusters.

These metrics show that the hybrid approach produces consistent clustering results. The clusters represent patterns of interaction between users and network services. Each cluster corresponds to a behavioral profile that summarizes typical activity within the dataset.

The clustering analysis confirms that dimensionality reduction improves the identification of behavioral patterns in user activity data. By combining representation learning with clustering, the method extracts structured behavioral profiles from the proxy logs. These profiles serve as the basis for the behavioral embeddings used in the following stages of the study.

The resulting behavioral dataset provides a representation of user activity patterns that can be integrated with network traffic data. These behavioral embeddings are later combined with network traffic features to support anomaly detection experiments.

3.6. Summary of Behavioral Profile

The results of the behavioral profiling stage show that user interaction patterns can be represented in a structured form using behavioral features and representation learning. The interaction frequencies capture how users distribute activity across services. The entropy values capture how concentrated or distributed the activity of each user is. Together with the principal components and the latent representations produced by the autoencoder, these variables provide a numerical description of user activity patterns.

The clustering analysis shows that dimensionality reduction improves the identification of behavioral structures in the dataset. When clustering algorithms are applied directly to the original behavioral features, the separation between user groups remains limited. In contrast, when the clustering process is applied to the latent representations generated by the autoencoder, the separation between clusters increases. The silhouette scores reported in Fig. 3.1 confirm that the hybrid approach produces more consistent behavioral groupings.

These results indicate that the latent space learned by the autoencoder captures relationships between behavioral variables that are not directly observable in the original feature space. The clustering process then groups users according to similarity in this latent representation. Each cluster therefore represents a behavioral profile that summarizes a typical pattern of interaction with network services.

The behavioral profiles obtained in this chapter provide a structured dataset that describes user activity patterns at the application level. Each profile represents the interaction structure of a user session and captures the distribution of activity across monitored services. This representation makes it possible to associate behavioral patterns with other forms of network data.

However, the behavioral dataset and the network traffic datasets used in anomaly detection experiments originate from different domains. The behavioral dataset describes user interaction patterns obtained from proxy logs, while the network traffic datasets describe packet-level or flow-level characteristics of network communication. These two sources of information capture different aspects of network activity.

For this reason, the next step of the methodology introduces a mechanism to integrate both sources of information. The objective is to associate behavioral embeddings with network traffic

records in order to provide contextual information about user activity patterns. This integration allows anomaly detection models to analyze network traffic not only through packet or flow features but also through behavioral context derived from user activity.

The following chapter introduces the cross-domain behavioral enrichment process used in this study. In that stage, behavioral embeddings obtained from the profiling process are aligned with network traffic datasets through representation learning techniques. This alignment produces enriched datasets that combine behavioral and traffic features in a shared representation space, enabling the development of anomaly detection models that account for both network activity and user behavior patterns.

3.7. Ethics, Consent, and Data Handling

Charles Proxy captures HTTP and HTTPS traffic, so the raw logs behind the behavioral dataset can contain URLs, IP addresses, cookies, timestamps, and session identifiers. Each of these fields can identify a person or a device, which places the dataset under the same handling requirements as any record of personal activity.

Participant consent was obtained before data collection began, as described in Section 3.3. What remains pending at the time of writing is formal approval or exemption from the corresponding institutional ethics committee, together with anonymization of URLs, IP addresses, and session identifiers [Sen-Dasgupta-23]. These steps depend on institutional offices outside the scope of a single pipeline run; their status at the time of submission is recorded in Table 3.5.

A technical basis for the dataset datasheet already exists, independent of the pending institutional steps. Table 3.5 summarizes what the diagnostic run over the raw records already established; provenance, population detail, limitations, and license terms will extend this table once committee approval and anonymization are completed.

TABLE 3.5

DATASHEET BASIS FOR THE BEHAVIORAL DATASET (CHARLES PROXY LOGS)

Field	Value
Users	150
Service categories	17
Duplicate rows	0
Missing values	0
Participant consent	Obtained
Ethics committee approval / exemption	Pending
Anonymization (URL, IP, session ID)	Pending

3.8. Published Work

The related work was published as:

Najar Juárez, C. S. (2023). *Towards a user network profiling for internal security top-k rankings similarity measures using K-means. In International Conference on Software Process Improvement (CIMPS 2023)*, Cuernavaca, Morelos, México.

Najar Juárez, C. S. (2024). *Towards a user network profiling for internal security top-k rankings similarity measures using K-means. In* J. Mejía, M. Muñoz, A. Rocha, Y. Hernández Pérez, & H. Avila-George (Eds.), *New perspectives in software engineering* (Studies in Computational Intelligence, Vol. 1135). Springer. https://doi.org/10.1007/978-3-031-50590-4_19

4. Cross-Domain Behavioral Enrichment for Network Traffic

4.1. Introduction

Cross-domain behavioral enrichment refers to the integration of information from different data domains in order to obtain a unified representation of network activity. In network security, traffic data describe communication patterns through flow-level features, while behavioral data describe patterns of interaction between users and services. These two sources capture different aspects of the same system. The integration of both domains is based on the assumption that network anomalies are related to deviations in both traffic structure and user activity. Previous studies have shown that anomaly detection improves when contextual information is incorporated into the analysis of network data [Salem-Stolfo-11; Hooshmand-Hosahalli-22; Singh-Jindal-23].

The enrichment process requires a mechanism to relate observations that do not share a direct correspondence. Traffic datasets and behavioral datasets are generated under different conditions and contain different feature spaces. For this reason, cross-domain methods focus on learning representations that allow comparison between domains. Representation learning techniques enable the projection of both sources into a common space, where relations between traffic patterns and behavioral patterns can be defined. This approach supports the construction of enriched datasets where each traffic observation is associated with behavioral context, which provides a basis for anomaly detection models that operate on integrated information.

The objective of this chapter is to integrate these behavioral representations with network traffic datasets used for anomaly detection. This integration creates a joint feature space that contains both traffic characteristics and user behavioral context. The resulting representation allows learning models to analyze network activity with additional contextual information about user behavior.

The chapter first describes the network traffic datasets used in this study, UNSW-NB15 and CIC-IDS2017, which contain labeled records of normal and malicious network traffic. The chapter

explains the structure of these datasets and the preparation steps required for their use in the enrichment process.

The chapter then presents the cross-domain enrichment methodology. This process aligns behavioral embeddings obtained from proxy traffic with network traffic records using adversarial autoencoder models. The adversarial training procedure produces latent representations that map behavioral data and network traffic into a shared feature space.

Once the latent spaces are aligned, each network traffic record is paired with the most similar behavioral embedding using cosine similarity. The selected behavioral representation is concatenated with the traffic features to produce an enriched feature vector. This enriched representation combines traffic characteristics with behavioral context.

The final section of the chapter summarizes the enrichment framework and describes how the enriched datasets are prepared for anomaly detection models. The enriched feature space produced in this chapter serves as the input for the deep learning models described in Chapter 5.

4.2. Background on Network Traffic and Anomalies

Network traffic is the exchange of data between devices connected to a communication network. Devices communicate by sending packets through network links using defined protocols. These protocols determine how data is transmitted, received, and interpreted between hosts. Common protocols include the Transmission Control Protocol and the User Datagram Protocol. TCP provides ordered delivery of packets, while UDP allows packet transmission without connection management [Cerf-Kahn-74].

The growth of network infrastructure has increased the volume of transmitted data. This growth requires systems that can monitor traffic and detect events that affect network operation. Monitoring systems analyze traffic flows in order to identify abnormal activity and protect network resources. These systems must process large volumes of traffic records generated by network communication [Hooshmand-Hosahalli-22].

Network anomaly detection refers to the process of identifying traffic patterns that differ

from normal activity. In many environments, anomalies indicate events such as intrusions, unauthorized access, or service misuse. Early intrusion detection systems relied on predefined rules or known attack signatures. These systems detect attacks that match stored patterns but cannot identify unknown threats or new variations of existing attacks [Martin-Fernandez-21].

Machine learning methods have been introduced to detect anomalies through statistical analysis of network data. These methods learn patterns of normal traffic from training data and identify deviations in new observations. Several models have been used for this task, including neural networks, convolutional networks, and recurrent neural networks. These models process features extracted from traffic records and learn patterns associated with network activity [Goodfellow-Bengio-Courville-16]. Network traffic records can be represented as numerical feature vectors that describe the properties of communication flows. These features may include packet size, protocol type, flow duration, byte counts, and packet timing. Let

$$X = \{x_1, x_2, ..., x_m\}$$

represent a set of network traffic observations where each vector

$$x_i \in \mathbb{R}^n$$

contains the features extracted from one traffic record. The objective of anomaly detection is to learn a function

$$f : \mathbb{R}^n \rightarrow \{0, 1\}$$

that assigns a label to each observation. The value

$$f(x) = 1$$

indicates anomalous traffic, while

$$f(x) = 0$$

indicates normal network activity [Hooshmand-Hosahalli-22].

Network anomalies appear in several forms depending on the attack objective. Denial of

Service attacks generate large volumes of traffic in order to exhaust system resources. Reconnaissance attacks scan network services to discover open ports and system configurations. Exploit attacks use vulnerabilities in software or communication protocols to gain system access. Malware activity may include worms, shellcode, or backdoors that execute commands on compromised hosts. These events produce deviations from normal traffic patterns and can be detected through analysis of network features [Wang-Balarezo-21].

Many anomaly detection models rely only on traffic-level features. These features describe packet behavior and flow statistics but do not include information about the user that generated the traffic. In shared network environments, traffic patterns often vary due to differences in user behavior. As a result, models trained only on traffic data may fail to capture contextual information related to user activity.

User behavior profiling has been studied as a complementary approach for network security analysis. This approach models patterns of user interaction with network services and applications. Behavioral profiles describe how users access platforms, services, or online resources. Changes in these patterns may indicate misuse, compromised accounts, or automated activity [Salem-Stolfo-11].

Behavioral analysis allows systems to distinguish normal user activity from suspicious behavior. By analyzing patterns of service access and activity distribution, security systems can detect deviations from typical usage patterns. This type of analysis supports early detection of malicious activity in network environments. The theoretical basis for measuring distributional variability in activity patterns draws from information theory [Shannon-48], and has been applied to network anomaly detection through entropy-based methods [Bashurov-Safonov-23].

Recent research shows that combining behavioral information with network traffic analysis can improve anomaly detection. Behavioral context provides additional information about how network services are used by individuals or groups. This information complements packet-level observations and supports the identification of patterns that may not be visible in traffic data alone [Hooshmand-Hosahalli-22].

In this study, network traffic analysis and user behavior profiling are combined into a

unified framework for anomaly detection. Network traffic records are represented as feature vectors extracted from communication flows. Behavioral embeddings derived from user activity logs provide contextual information about service usage patterns. The integration of these two sources of information supports the identification of anomalous traffic patterns. The following section describes the datasets used in this study and the preparation process applied before the enrichment stage.

4.3. Overview of Network Traffic Datasets

Network anomaly detection requires datasets that contain records of network activity with labels that indicate normal or malicious behavior. These datasets allow the training and evaluation of detection models. Each record represents a communication event described by numerical features extracted from network traffic flows.

In this study, two benchmark datasets are used for anomaly detection experiments. These datasets are UNSW-NB15 and CIC-IDS2017. Both datasets contain labeled traffic records that represent normal network activity and different types of attacks. Using more than one dataset allows the evaluation of detection models under different traffic distributions and network conditions.

The datasets are used as the traffic domain in the cross-domain enrichment framework described in this work. Each traffic record is later associated with behavioral embeddings obtained from user activity data. This integration allows the learning model to analyze traffic characteristics together with contextual behavioral information. The following subsections describe the structure and preparation of the datasets used in this study.

4.4. Network Traffic Datasets

4.4.1. UNSW-NB15 Dataset

The UNSW-NB15 dataset is a network intrusion detection dataset developed at the Cyber Range Lab of the Australian Centre for Cyber Security. The dataset was generated using a hybrid network

simulation environment where normal traffic and attack traffic were captured and labeled. The dataset was designed to represent modern network activity and contemporary attack scenarios [Moustafa-Slay-15]. The dataset is publicly available through the Australian Centre for Cyber Security at https://research.unsw.edu.au/projects/unsw-nb15-dataset

The dataset contains network flow records generated from packet captures. Each record describes a communication flow between two hosts. Flow features include statistical measurements extracted from packets observed during the communication session. These features describe properties of the traffic such as duration, packet counts, byte counts, and protocol information.

Let $X = \{x_1, x_2, ..., x_m\}$ be the set of network traffic records contained in the dataset. Each traffic record is represented as a feature vector $x_i \in \mathbb{R}^n$ where n corresponds to the number of extracted traffic features. Each record also contains a label $y_i \in \{0, 1\}$ that indicates whether the traffic instance corresponds to normal activity or an attack. The objective of anomaly detection models is to learn a function $f(x_i) = y_i$ that maps traffic features to their corresponding class labels.

The UNSW-NB15 dataset includes several categories of attacks that represent common threat scenarios observed in network environments. These attacks include:

- Denial of Service attacks
- Reconnaissance attacks
- Exploit attacks
- Generic attacks
- Shellcode attacks
- Worm propagation attacks

These attack categories represent different strategies used by attackers to disrupt services, collect information about systems, or gain unauthorized access.

In this study, the binary version of the dataset is used. All attack categories are grouped into a single class labeled as malicious traffic. This configuration produces a binary classification task where each record belongs to either the normal class or the attack class. The binary setting

simplifies the learning problem and allows the enrichment process to associate behavioral context with each traffic instance without requiring semantic correspondence between specific attack types and user activity patterns.

Before training the detection models, several preprocessing steps are applied to the dataset. Features that contain direct identifiers such as user identifiers or IP addresses are removed in order to avoid bias in the learning process. The remaining numerical features are retained for model training. The dataset is then divided into training and testing subsets using a 70/30 split. Let X_{train} represent the set of training samples and X_{test} represent the set of testing samples. This separation ensures that the model is evaluated on data that were not used during training.

The UNSW-NB15 dataset provides a large collection of traffic flows that include both normal network behavior and multiple forms of malicious activity. These characteristics make the dataset suitable for evaluating anomaly detection models that operate on network traffic features.

4.4.2. CIC-IDS2017 Dataset

The CIC-IDS2017 dataset is another benchmark dataset used for network intrusion detection research. The dataset was developed by the Canadian Institute for Cybersecurity in order to provide realistic traffic data that include both benign activity and attack scenarios [Rosay-Cheval-22]. The dataset is publicly available through the Canadian Institute for Cybersecurity at https://www.unb.ca/cic/datasets/ids-2017.html

The dataset was generated by simulating normal user activity and multiple types of network attacks within a controlled network environment. Traffic was captured over several days and labeled according to the attack events performed during the experiment.

Each traffic record in the dataset corresponds to a network flow extracted from packet captures. Flow features describe properties of the communication session, including packet statistics, timing information, and protocol attributes.

As with the UNSW-NB15 dataset, each flow can be represented as a feature vector $x_i \in \mathbb{R}^n$ where n corresponds to the number of traffic features extracted from the flow. Each record is associated with a label $y_i \in \{0, 1\}$ that indicates whether the traffic corresponds to benign

activity or malicious activity.

The CIC-IDS2017 dataset contains several attack categories that simulate real network threats. These categories include:

- Distributed Denial of Service attacks
- Brute force attacks
- Web attacks
- Botnet traffic
- Port scanning activity
- Infiltration attacks

These attack types represent different strategies used by attackers to compromise network systems or disrupt services.

In this study, the dataset is prepared using a stratified subsampling process. A representative subset corresponding to approximately five percent of the dataset is extracted while preserving the original distribution between normal and malicious samples. This approach reduces computational cost while maintaining the statistical structure of the dataset. Before this subsampling step, 143,521 exact duplicate rows (12.35% of the raw CIC-IDS2017 file) are removed so that identical records cannot be split across the training and test partitions.

Only numerical traffic features are retained for the experiments. Non-numerical attributes are removed. Missing values and undefined values are replaced using the median value of the corresponding feature. This process ensures that the dataset does not contain undefined inputs that could affect model training.

After preprocessing, the features are standardized to ensure that each variable contributes equally during model learning. Standardization transforms each feature using the following expression:

$$x_i' = \frac{x_i - \mu}{\sigma}.$$

where x_i represents the original feature value, μ represents the mean of the feature, and σ represents

the standard deviation. The resulting dataset contains normalized numerical feature vectors that represent network communication flows.

4.5. Cross-Domain Behavioral Enrichment

The behavioral profiles described in Chapter 3 represent patterns of user interaction with network services. Network anomaly detection datasets represent traffic flows generated by communication between devices. These two sources describe different aspects of network activity. Behavioral profiles describe user interaction patterns. Traffic datasets describe packet and flow characteristics.

In this study, both sources are integrated through a cross-domain enrichment process. The objective of this process is to associate each network traffic record with a behavioral representation obtained from user activity data. This integration allows anomaly detection models to analyze traffic characteristics together with user behavior patterns.

The enrichment process includes three steps. First, latent representations of both datasets are aligned using adversarial learning. Second, the most similar behavioral embedding is selected for each traffic record using cosine similarity. Third, the selected behavioral embedding is concatenated with the traffic representation to construct the enriched dataset.

4.5.1. Adversarial Alignment

The behavioral dataset and the network traffic datasets contain features that describe different phenomena. Behavioral data represent the frequency of interaction between users and services. Traffic datasets represent properties of network flows such as packet statistics and protocol attributes. Because these datasets describe different aspects of network activity, their feature distributions are not directly comparable.

The traffic dataset is split into training, validation, and test partitions before this alignment starts, and only the training partition feeds the autoencoders below. Validation and test embeddings are produced afterward by passing each partition through the already-trained encoder, never by fitting on them.

To address this problem, a latent representation is learned for each dataset using

autoencoders. Two independent autoencoder models are defined. One autoencoder processes the traffic dataset. The second autoencoder processes the behavioral dataset obtained from the Charles Proxy logs.

Each autoencoder contains an encoder and a decoder. The encoder maps the input feature vector into a latent representation. The decoder reconstructs the input vector from this latent representation. During training, the model minimizes the difference between the original input and the reconstructed output.

The reconstruction objective is defined using the mean squared error loss function. For an input vector x_i, the reconstruction error is computed as

$$L_{rec} = \|x_i - \hat{x}_i\|^2.$$

where x_i is the original input vector and $\hat{x}_i$ is the reconstructed output produced by the decoder.

In addition to the reconstruction objective, a discriminator network is introduced. The discriminator receives latent vectors generated by the two autoencoders and attempts to determine their source domain. The discriminator performs binary classification to determine whether an embedding originates from the traffic dataset or from the behavioral dataset. The autoencoders and the discriminator are trained simultaneously through an adversarial process. During training, the discriminator learns to correctly classify the origin of the latent embeddings. At the same time, the autoencoders attempt to produce latent representations that prevent the discriminator from distinguishing between the two domains.

This adversarial process forces the latent representations produced by the two autoencoders to follow similar distributions. As training progresses, the embeddings from the traffic dataset and the behavioral dataset become aligned within a shared latent space. This shared representation space enables the integration of both domains without requiring direct correspondence between their original features.

The training configuration used for this adversarial alignment is defined by a set of parameters that control the optimization process and the latent representation size. These parameters are summarized in Table 4.1. The adversarial autoencoder training uses a batch size of

128 samples and a latent representation of dimension 32. The learning rate used for optimization is 0.001.

TABLE 4.1

TRAINING PARAMETERS USED IN ADVERSARIAL AUTOENCODER DATASET ENRICHMENT

Parameter	Value
Batch size	128
Latent dimension	32
Learning rate	0.001
Number of epochs	50
Reconstruction loss	Mean Squared Error
Adversarial loss weight	0.1
Optimizer	Adam

Training runs for 50 epochs using the Adam optimizer, the same number for UNSW-NB15 and for CIC-IDS2017. An earlier version of the code used 50 epochs for UNSW-NB15 and 20 for CIC-IDS2017 with no stated reason; the corrected pipeline fixes one shared value across both datasets. The reconstruction loss is computed using mean squared error. The adversarial component of the loss is computed using binary cross-entropy and is weighted by a factor of 0.1 relative to the reconstruction loss.

The encoder and decoder architectures used during this process correspond to the configuration described previously for the behavioral representation model. The encoder transforms the input feature vector into a latent representation of dimension 32. The decoder reconstructs the original feature vector from this representation. The discriminator receives the latent embeddings and performs binary classification between the two domains.

4.5.2. Cosine Similarity Matching

After adversarial training, latent embeddings are extracted from the encoder of each

autoencoder model. Each network traffic record is represented by a latent vector produced by the traffic encoder. Each behavioral profile is represented by a latent vector produced by the behavioral encoder.

Let u_i denote the latent representation of the i-th network traffic record. Let c_j denote the latent representation of the j-th behavioral profile. The similarity between these two vectors is measured using cosine similarity. Cosine similarity measures the angle between two vectors in the latent space. The similarity score is computed as

$$sim(u_i, c_j) = \frac{u_i \cdot c_j}{\|u_i\|\|c_j\|},$$

where $u_i \cdot c_j$ denotes the dot product between the two vectors and $\|u_i\|$ and $\|c_j\|$ denote the Euclidean norms of the vectors.

For each traffic embedding u_i, the similarity value is computed with every behavioral embedding c_j. The behavioral embedding that produces the maximum similarity score is selected. This embedding corresponds to the behavioral profile that most closely matches the traffic representation in the latent space.

This procedure produces an instance-level association between traffic data and behavioral profiles. Each network traffic record is paired with the behavioral representation that has the highest similarity score.

4.5.3. Semantic Validity of the Behavioral Match

The behavioral profiles do not come from the same users who generated the traffic in UNSW-NB15 or CIC-IDS2017, so a high cosine similarity between a traffic embedding and a behavioral embedding does not by itself show a meaningful relation. It could just as well reflect closeness that carries no interpretable content.

To check this, the cosine similarity produced by the proposed match was compared against the similarity produced by a random pairing, using a Mann-Whitney U test over thousands of pairs. In UNSW-NB15, the proposed match reached a mean similarity of 0.28 against 0.036 for the random pairing, with p below 0.000001. CIC-IDS2017 repeated the pattern, with 0.31 against

0.051 at the same significance level.

This result rules out one narrow claim and leaves another open. It shows that the match is not mathematically equivalent to a random pairing. It does not show that a behavioral profile and the traffic record paired with it share a real correspondence, since, as noted above, they come from different users and different contexts. The enrichment framework is therefore presented in this thesis as a synthetic enrichment strategy, not as evidence of a causal link between a specific user's behavior and a specific traffic flow.

4.5.4. Enriched Dataset Construction

After the matching step, each network traffic embedding is associated with a behavioral embedding selected through cosine similarity. These two representations are combined in order to construct the enriched dataset. Let u_i be the latent representation of the traffic record and c_j be the selected behavioral embedding. The enriched representation is defined as

$$\tilde{u}_i = [u_i; c_j], \qquad \text{Eq. 5}$$

where the operator $[\cdot;\cdot]$ denotes vector concatenation. This operation produces a joint feature vector that integrates information from both domains. The traffic embedding represents packet and flow characteristics extracted from the network dataset. The behavioral embedding represents interaction patterns derived from user activity logs. The resulting enriched vector preserves the original label of the traffic dataset. The enrichment process therefore does not modify the classification target. Instead, it expands the feature space by incorporating behavioral information associated with each network record. The enriched dataset created through this process is used as the input for the anomaly detection model described in the next chapter.

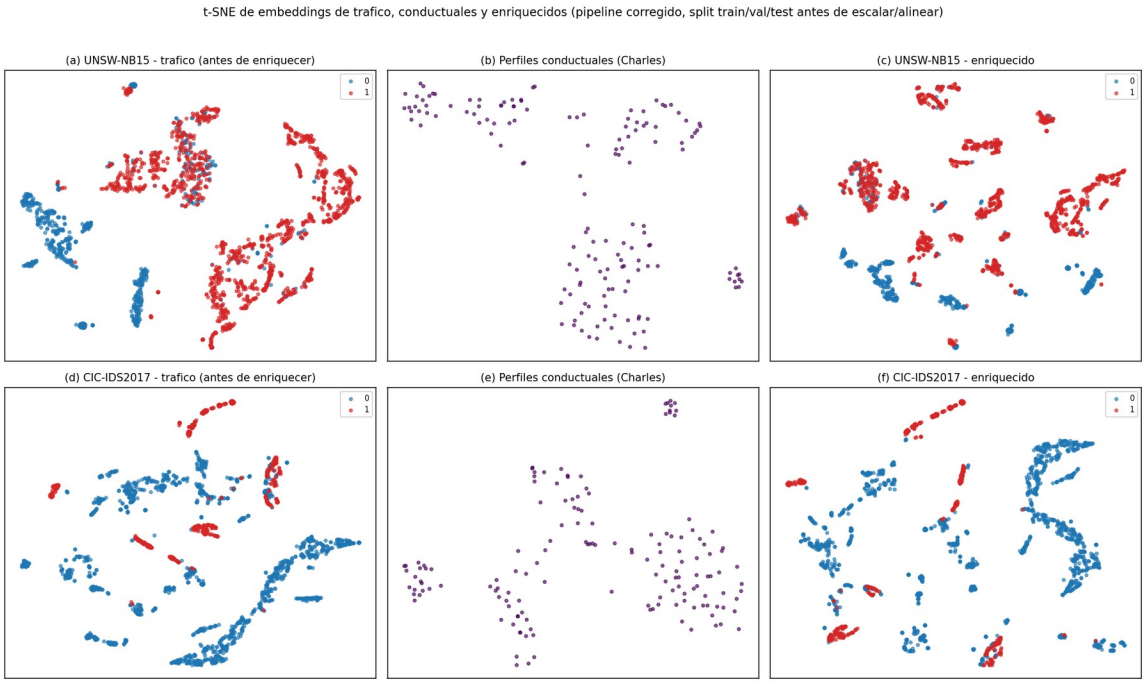


Fig. 4.1 t-SNE visualization of embeddings from the leakage-free pipeline (split before scaling, autoencoder training, and adversarial alignment). (a) UNSW-NB15 traffic, test partition, colored by class. (b) Charles behavioral profiles. (c) UNSW-NB15 enriched. (d)-(f) same views for CIC-IDS2017.

4.5.5. Ablation Study Across Enrichment Variants

A behavioral embedding concatenated to a traffic embedding could raise accuracy for a reason that has nothing to do with behavior: added dimensions alone can shift a classifier's decision surface. The variants below isolate that possibility by replacing the real behavioral embedding with controls of identical dimensionality: a zero vector, Gaussian noise, a randomly assigned behavioral embedding, and a permuted pairing that keeps every embedding but scrambles which traffic record receives which profile. Two further variants remove a mechanism instead of the content: one drops the adversarial discriminator and trains both autoencoders independently before matching by cosine similarity, and one drops PCA from the behavioral profile. A last group varies the latent dimension across 16, 32, and 64. Every variant uses the same split, the same seed, and the same classifier architecture as the proposed method.

Table 4.2 reports the result for UNSW-NB15. None of the enrichment variants, including the real behavioral embedding, moves accuracy beyond what a synthetic control already reaches. Fig. 4.2 plots the same comparison: accuracy and ROC AUC for every variant sit within a narrow band around the no-enrichment baseline (dashed line), with no variant, including the real

embedding, pulling clearly above it.

TABLE 4.2

ABLATION RESULTS ON UNSW-NB15 (SAME SPLIT, SEED, AND CLASSIFIER ACROSS VARIANTS)

Variant	Accuracy
No enrichment (baseline)	0.9328
Real behavioral embedding	0.9352
Zero vector	0.9327
Gaussian noise	0.9273
Random assignment	0.9325
Permuted pairing	0.9334

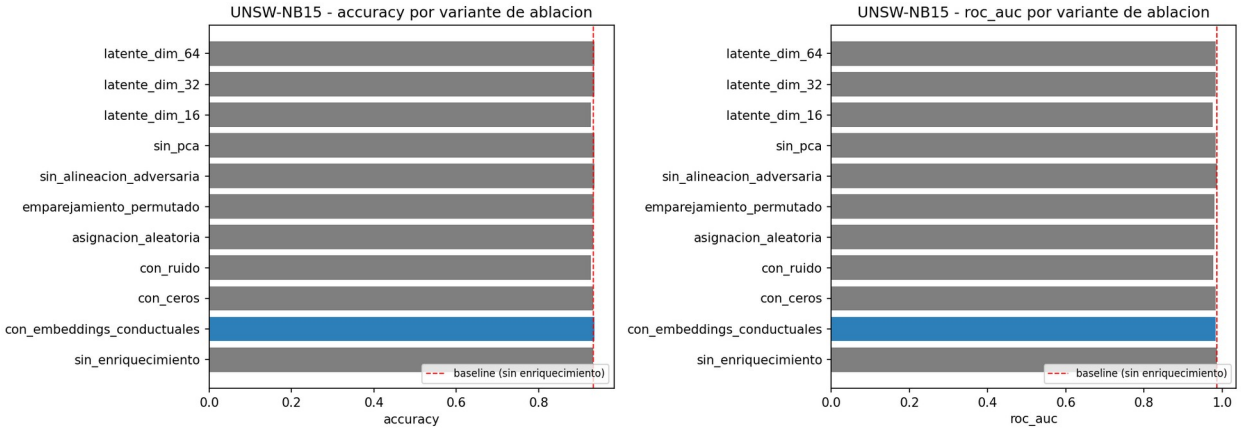


Fig. 4.2 Accuracy and ROC AUC for every ablation variant on UNSW-NB15. The dashed line marks the no-enrichment baseline; every bar sits close to it.

CIC-IDS2017 repeats the same pattern with a wider margin. The no-enrichment baseline reaches 0.9598, and nearly every enriched variant, including the real behavioral embedding (0.9539), lands at or below that number, as Fig. 4.3 shows; the sole exception is the no-adversarial-alignment variant, which reaches 0.9612. Removing the adversarial discriminator, dropping PCA from the behavioral profile, and moving the latent dimension across 16, 32, and 64 produced the same outcome in both datasets: none of these changes lets the real embedding pull ahead of its controls.

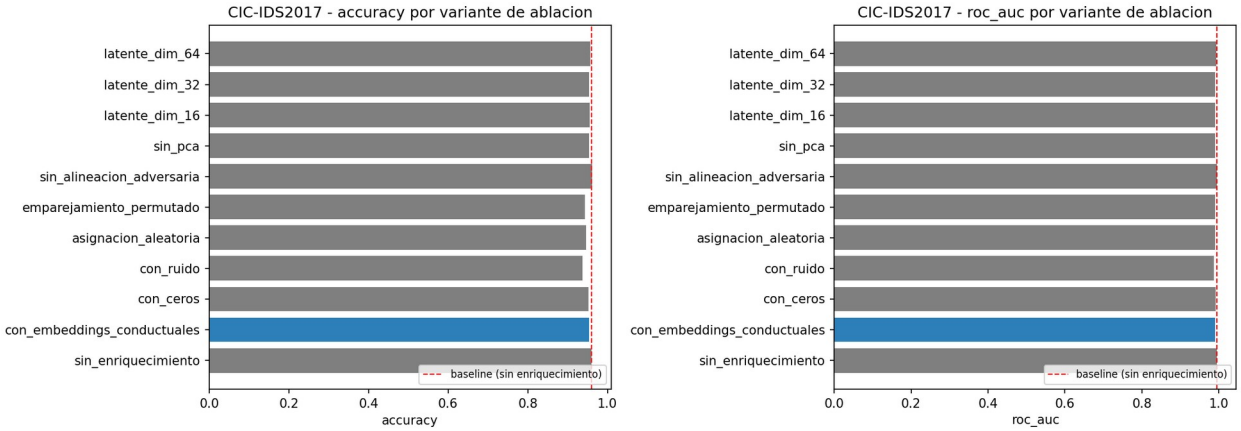


Fig. 4.3 Accuracy and ROC AUC for every ablation variant on CIC-IDS2017. Nearly every enriched variant, including the real embedding, falls at or below the no-enrichment baseline; the only exception is the variant without adversarial alignment.

Applied across the full dataset, behavioral enrichment therefore adds dimensionality without adding signal that a synthetic control could not already supply. Chapter 5 returns to this same set of controls under a different combination rule, a confidence cascade that restricts enrichment to a narrow band of the traffic classifier's output, and finds a different outcome there.

4.6. Enriched Feature Space

The enrichment process produces a joint representation that combines network traffic embeddings with behavioral embeddings. Each traffic record is represented by a latent vector obtained from the traffic autoencoder. Each behavioral profile is represented by a latent vector obtained from the behavioral autoencoder. These vectors are concatenated as defined in Eq. 5. The result is a single vector that contains information from both domains.

The enriched representation expands the feature space of the traffic dataset. The traffic component captures statistical properties of network flows. The behavioral component captures patterns of user interaction with monitored services. The concatenation of these vectors creates a representation that contains both traffic activity and behavioral context.

The structure of the enriched feature space can be observed in the embedding visualization shown in Fig. 4.1. The figure presents a two-dimensional projection of the latent representations obtained through t-SNE [Marx-24]. The projection shows the spatial distribution of embeddings

produced by the enrichment process.

Sub-figures (a) and (d) of Fig. 4.1 show the embeddings produced from the UNSW-NB15 and CIC-IDS2017 traffic datasets before enrichment, respectively. Traffic samples form groups that reflect patterns present in the flow statistics. Normal traffic and attack traffic occupy different regions of the projection space.

Sub-figures (b) and (e) of Fig. 4.1 show the embeddings obtained from the behavioral dataset derived from proxy logs. Behavioral profiles appear as groups that represent patterns of interaction with monitored services. These groups correspond to clusters obtained during the behavioral profiling process described in Chapter 3.

Sub-figures (c) and (f) of Fig. 4.1 show the embeddings after the enrichment step. In this representation, traffic embeddings are combined with behavioral embeddings through vector concatenation. The projection shows a visible separation between normal and anomalous traffic samples. The enriched representation forms clusters that reflect both traffic behavior and behavioral context.

The enriched space therefore preserves structural information from both domains. Traffic patterns remain visible in the projection, while behavioral patterns introduce additional structure into the representation. This combined structure supports the identification of anomalous activity within the traffic dataset.

The visualization shown in Figure 4.1 illustrates that the enriched embeddings form groups that correspond to patterns of network activity. Normal traffic samples appear concentrated in specific regions of the latent space. Attack samples appear separated from these regions. This separation is consistent with the enriched representation preserving the information already present in the traffic features.

A t-SNE projection is, however, a qualitative view. The ablation study in Section 4.5.5 shows that classification on this space performs no better than on spaces where the behavioral embedding is replaced by synthetic controls, so the separation visible in Fig. 4.1 is better attributed to the traffic component than to behavioral context. The enriched feature space created in this chapter is used as the input representation for the anomaly detection models described in Chapter 5.

4.7. Summary of the Enrichment Framework

The enrichment process combines two sources of information about network activity. One source describes traffic flows observed in communication networks. The other source describes patterns of interaction between users and online services. Each source captures a different aspect of network behavior.

Traffic datasets describe the structure of communication flows. They contain measurements derived from packets and connections between hosts. These measurements allow the identification of patterns related to network activity and malicious behavior. However, traffic features do not describe how users interact with services or how activity is distributed across platforms.

Behavioral profiles describe patterns of user interaction with network services. These profiles summarize how users access platforms, generate requests, and distribute activity across services. Behavioral information therefore provides context about how network resources are used.

The integration of both sources produces a representation that contains traffic characteristics and behavioral context. This representation allows the learning process to observe patterns that are not visible when each source is analyzed independently. Traffic records are no longer interpreted only as isolated flows. They are associated with behavioral patterns that reflect how users interact with network services.

This association introduces contextual information into the representation of network activity. Traffic patterns that appear similar at the packet level may correspond to different forms of user behavior. By incorporating behavioral context, the framework is designed to let the learning model distinguish between patterns that belong to normal usage and patterns that indicate abnormal activity; whether it does so is tested by the ablation study above and in Chapter 5.

The enrichment framework also addresses a limitation present in many network datasets. Traffic datasets contain labels that describe attack activity but do not include information about the users that generate the traffic. Behavioral data provides an additional layer of information that complements the structure of the traffic dataset.

Another implication of this approach relates to representation learning. The alignment of behavioral and traffic embeddings allows both domains to be analyzed within a shared representation space. This shared space enables the comparison of patterns that originate from different sources of data. The resulting representation may contain relationships that are not present in the original feature spaces.

The enriched dataset produced in this chapter therefore represents a combination of network observations and behavioral patterns. This representation expands the information available for anomaly detection.

Two claims need separating at this point. The autoencoders, the adversarial alignment between domains, and the cosine similarity used for matching all come from established work in representation learning; this thesis applies them rather than inventing them. What this thesis contributes is the experimental frame built around them: the full set of ablation controls above, and the confidence cascade introduced in Chapter 5, which restricts enrichment to traffic records the base classifier cannot yet resolve. Of four combination rules tested, the confidence cascade is the only one for which the evidence in Chapter 5 holds up under repetition.

Because the ablation results above rule out a general benefit from behavioral enrichment, the framework in this thesis is presented as a synthetic enrichment or regularization strategy rather than as a demonstration that user behavior improves traffic-based detection on its own. The Charles Proxy dataset remains a contribution of this work, but as a resource other studies can reuse, not as proof that pairing it with traffic records produces a working detector by itself.

Chapter 5 examines how this enriched representation is used to train deep learning models for network anomaly detection. These models learn patterns of network activity from the enriched feature space and evaluate whether the integration of behavioral context improves the identification of anomalous traffic.

5. Deep Learning Models for Network Anomaly Detection

5.1. Introduction

Network anomaly detection aims to identify patterns of network activity that differ from normal behavior. Machine learning models analyze traffic records in order to detect these deviations. Deep learning models have been used for this task because they can learn complex patterns from large datasets [Goodfellow-Bengio-Courville-16].

The previous chapter introduced a method to integrate behavioral profiles with network traffic data. Behavioral embeddings derived from user activity logs were aligned with network traffic representations. This process produced an enriched dataset where each traffic record contains both traffic information and behavioral context.

The enriched feature space combines two sources of information about network activity. Traffic features describe properties of communication flows. Behavioral embeddings describe patterns of user interaction with network services. The resulting representation provides a joint description of network behavior.

This chapter examines how anomaly detection models learn from this enriched representation. Deep learning models are trained to classify network traffic records as normal activity or anomalous activity. The models process feature vectors that represent network flows and behavioral context.

The chapter first describes the representation of the input features used for anomaly detection. Two configurations are considered. The first configuration uses only network traffic features. The second configuration uses enriched feature vectors that combine traffic features with behavioral embeddings.

The chapter then presents the architecture of the deep feedforward neural network used for anomaly detection. The training procedure is described together with the parameters used during

model optimization. The evaluation framework is also introduced. This framework defines the metrics used to measure the performance of the detection models.

Finally, the chapter presents the experimental results obtained from the trained models. The results compare models trained with traffic features only and models trained with enriched feature vectors. This comparison allows the analysis of the effect of behavioral context on anomaly detection performance.

5.2. Input Representation for Anomaly Detection

This section describes the representation of the data used by the anomaly detection model. The input vectors come from the network traffic datasets and from the enriched representation created in the previous chapter.

Network traffic records are represented as numerical feature vectors extracted from communication flows. Each vector describes properties of the traffic session such as packet statistics, protocol attributes, and flow measurements. These vectors are used as input to the deep learning model that performs anomaly classification.

Two input configurations are evaluated in this study. The first configuration uses only traffic features extracted from the network datasets. The second configuration uses enriched feature vectors that integrate network traffic information with behavioral embeddings obtained from user activity logs.

This comparison allows the analysis of the effect of behavioral context on anomaly detection performance. The representations described in this section are used as input to the deep feedforward neural network presented later in the chapter.

The complete data preparation process that produces these representations is described in Section 1.5 and Chapter 4.

5.2.1. Traffic Feature Representation

The first representation corresponds to the baseline configuration used for comparison. In this

setting, the anomaly detection model is trained using only network traffic features. Network traffic records are extracted from the benchmark datasets used in the study. These datasets include the UNSW-NB15 dataset and the CIC-IDS2017 dataset. Each record corresponds to a network flow described by a set of numerical features extracted from packet captures.

Let $x_i \in \mathbb{R}^n$ represent a network traffic observation. The vector x_i contains the features extracted from one communication flow. These features include measurements related to packet counts, byte counts, protocol information, and temporal properties of the connection.

Each traffic record is associated with a binary label $y_i \in \{0, 1\}$ where the value $y_i = 0$ represents normal network activity and $y_i = 1$ represents malicious traffic. The anomaly detection model learns a mapping between the traffic feature vector and the corresponding class label.

The baseline configuration uses only the traffic feature vector as input to the classification model. No behavioral information is included in this representation. The objective of this configuration is to provide a reference model that reflects the performance obtained when anomaly detection relies exclusively on network traffic parameters.

This baseline configuration allows a direct comparison with the enriched representation described in the following subsection. The comparison makes it possible to evaluate whether the integration of behavioral context modifies the structure of the feature space and the learning dynamics of the anomaly detection model.

5.2.2. Enriched Behavioral-Traffic Representation

The second representation corresponds to the enriched feature space generated through the cross-domain alignment process. In this configuration, the input vector combines network traffic information with behavioral embeddings derived from user activity data.

The enrichment process aligns the latent representations of the traffic datasets and the behavioral dataset using adversarial autoencoder training. This alignment maps both domains into a shared latent space where embeddings from each dataset follow similar distributions.

After alignment, each traffic embedding is associated with a behavioral embedding selected through cosine similarity. This association links each network traffic record with the

behavioral profile that presents the highest similarity in the latent space.

Let u_i represent the latent representation produced by the encoder for a network traffic record. Let c_j represent the behavioral embedding obtained from the behavioral dataset. The enriched feature vector is defined as the concatenation of these two embeddings:

$$\tilde{u}_i = [u_i; c_j].$$

This operation produces a joint vector that contains information from both domains. The traffic embedding represents flow-level network characteristics. The behavioral embedding represents patterns of interaction with online services obtained from proxy traffic logs. The concatenated representation forms the final input vector used by the anomaly detection model.

The dimensionality of the enriched feature space corresponds to the combination of the traffic latent representation and the behavioral embedding.

TABLE 5.1

ARCHITECTURE AND TRAINING PARAMETERS OF THE DEEP FEEDFORWARD NEURAL NETWORK CLASSIFIER

Parameter	Value
Input dimension	96
Number of hidden layers	3
Hidden layer sizes	256, 128, 64
Activation functions	LeakyReLU, slope 0.1 (hidden layers), Sigmoid (output layer)
Dropout rate	0.3 (after each hidden layer)
Loss function	Binary Cross-Entropy
Optimizer	Adam
Learning rate	1e-4 (weight decay 1e-5)
Batch size	128
Number of epochs	up to 50, early stopping on validation loss (patience 7)

The deep learning classifier used in this study receives an input vector with 96 features, composed of:

- a 64-dimensional latent representation derived from the network traffic dataset
- a 32-dimensional behavioral embedding derived from the Charles Proxy behavioral dataset

This structure is described in the architecture of the Deep Feedforward Neural Network reported in Table 5.1. The enriched representation therefore expands the feature space available to the classifier. The model receives information about packet-level communication patterns together with behavioral context associated with user activity. The integration of these two sources of information creates a joint feature representation that links traffic observations with behavioral patterns observed in network usage.

Table 5.1 above reflects the classifier as the corrected code trains it. An earlier version of this table listed hidden layer sizes of 128, 64, and 32 with a plain ReLU activation and 20 training epochs; the network actually trains three hidden layers of 256, 128, and 64 units with a leaky ReLU activation, a learning rate of 1e-4 with weight decay, and early stopping that can run for up to 50 epochs. This same architecture, unchanged, is what this chapter trains for both the baseline and the enriched classifier.

5.3. Neural Network Architecture

The anomaly detection model is implemented as a Deep Feedforward Neural Network. Feedforward neural networks process input vectors through a sequence of transformation layers. Each layer performs a linear transformation followed by a nonlinear activation function [Goodfellow-Bengio-Courville-16].

Let $z \in \mathbb{R}^d$ represent the enriched feature vector obtained from traffic and behavioral data. The neural network computes a mapping $f : \mathbb{R}^d \to \{0, 1\}$ where the output represents the predicted class label for the traffic instance.

The architecture consists of an input layer followed by three hidden layers and one output layer. The input layer receives the enriched feature vector. The dimension of the input corresponds to the number of features contained in the combined representation.

Each hidden layer performs a transformation defined by $h_l = \sigma(W_l h_{l-1} + b_l)$ where W_l represents the weight matrix of layer *l*, b_l represents the bias vector, σ represents the activation function, and h_l represents the output of the layer.

The hidden layers use the leaky ReLU activation function with a negative slope of 0.1, as listed in Table 5.1. This function introduces nonlinearity into the network and allows the model to learn complex relationships between input variables [Goodfellow-Bengio-Courville-16].

The output layer contains a single neuron that produces the probability of the anomalous class. The sigmoid activation function maps the output value to the interval $[0, 1]$. The predicted class label is defined as $\hat{y} = 1$ if $p \geq 0.5$ and $\hat{y} = 0$ if $p < 0.5$ where p represents the predicted

probability produced by the network.

This architecture allows the neural network to learn decision boundaries that separate normal traffic from anomalous traffic in the enriched feature space.

5.4. Training Strategy

The neural network parameters are learned through supervised training. The training dataset contains labeled feature vectors that represent normal traffic and anomalous traffic. Let $D = \{(z_1, y_1), (z_2, y_2), \ldots, (z_n, y_n)\}$ represent the training dataset where z_i represents the enriched feature vector and $y_i \in \{0, 1\}$ represents the class label. The objective of training is to minimize the loss function over the dataset. Binary cross-entropy is used as the loss function:

$$L = -\sum [y \log(p) + (1 - y) \log(1 - p)].$$

where p represents the predicted probability produced by the network.

The parameters of the neural network are optimized using the Adam optimization algorithm. Adam computes adaptive learning rates for each parameter using estimates of first and second moments of the gradients [Goodfellow-Bengio-Courville-16].

Training proceeds in mini-batches. For each batch, the network computes predictions, evaluates the loss function, and updates the parameters through gradient descent. Regularization techniques are applied during training to reduce the risk of overfitting. Dropout layers randomly deactivate a subset of neurons during each training iteration. This mechanism prevents the network from relying on specific neurons and encourages the learning of distributed representations. Training runs for up to 50 epochs. After each epoch, the model is evaluated on the validation set, and training stops early when the validation loss does not improve for seven consecutive epochs (Table 5.1).

Training was performed on Google Colab using an NVIDIA Tesla T4 GPU with 16 GB of memory. The complete implementation is provided in Appendix B.

5.5. Evaluation Framework

The performance of the anomaly detection model is evaluated using several classification metrics. These metrics measure the ability of the model to correctly classify traffic instances. Let TP represent true positives, TN represent true negatives, FP represent false positives, and FN represent false negatives. Accuracy measures the proportion of correctly classified instances:

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

Precision measures the proportion of predicted anomalies that correspond to actual anomalies:

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

Recall measures the proportion of actual anomalies that are correctly detected:

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

The F1-score combines precision and recall into a single measure:

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall}$$

These metrics provide complementary information about classifier performance. Accuracy reflects overall performance, precision measures the rate of false alarms, recall measures detection capability, and the F1-score balances both measures.

Accuracy, precision, recall, and F1-score say little on their own about a classifier evaluated on imbalanced traffic, so five further metrics are reported for every run below: PR-AUC, the Matthews correlation coefficient, balanced accuracy, specificity, and the Brier score as a calibration check [Muntean-Militaru-23; Shirdel-DiMauro-24]. Every comparison between a baseline and an enriched model also carries a paired t-test, a bootstrap of the difference over 10,000 resamples, and a 95% confidence interval computed across multiple seeds, so that a gap between

two numbers can be read against a measure of how much that gap could arise from seed variation alone.

Cross-validation is used to evaluate the stability of the model across multiple training and validation splits. This procedure reduces dependence on a single partition of the dataset and provides a more reliable estimate of model performance. The evaluation framework allows the comparison between models trained on traffic-only features and models trained on enriched representations that include behavioral embeddings.

5.6. Classification Results

This section presents the classification results of the anomaly detection model. The evaluation includes two datasets: UNSW-NB15 and CIC-IDS2017. Two model settings are compared. The first is a baseline model that uses the original network traffic data. The second is the proposed model that uses traffic data with enriched behavioral information. The results show the performance of both models and allow comparison between the baseline approach and the proposed method with enriched data.

5.6.1. Results on the UNSW-NB15 Dataset

This section reports the classification results obtained on the UNSW-NB15 dataset. Two training settings are studied. The first setting uses the original traffic dataset. The second setting uses traffic data with behavioral enrichment.

The enrichment step integrates user activity information derived from proxy interaction logs with the network traffic records. The purpose of this comparison is to examine how the enriched representation influences the training process and the classification outcome.

Fig. 5.1 shows the learning curves for both models. In Sub-figure 5.1(a) the model uses the enriched dataset. The training process progresses across fifty epochs. The training loss starts near 0.67 and decreases during the training process. The validation loss follows a similar pattern and stabilizes near 0.05. A gap between the two curves remains during the entire training stage.

The loss curves in Fig. 5.1 come from the pipeline as it stood before the leakage fix

described below. They are kept here only to show how training and validation loss moved during optimization; every accuracy, confusion matrix, and ROC value reported from this point on uses the corrected pipeline, in which the scaler, the autoencoders, the adversarial alignment, and class balancing are fit on the training partition alone, never on validation or test data.

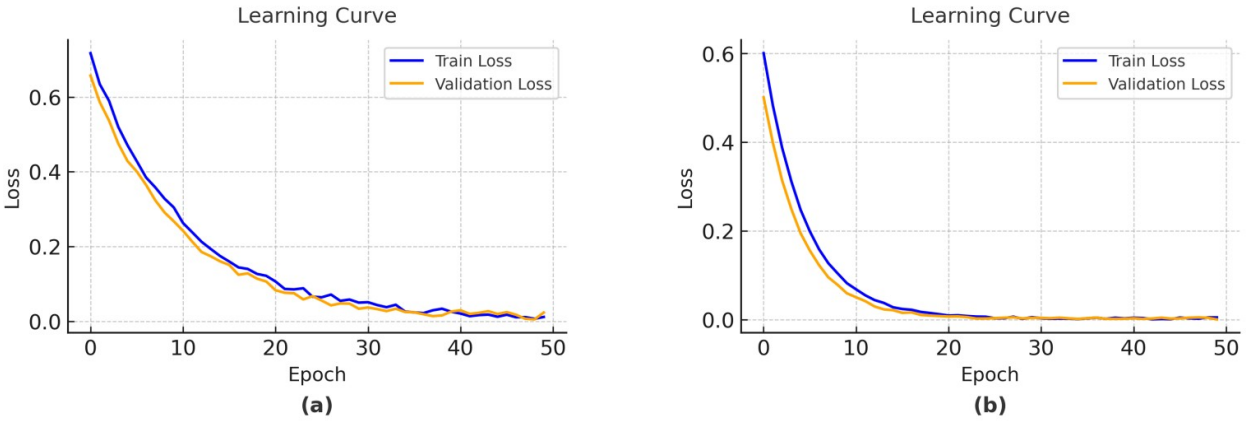


Fig. 5.1 Learning curves for DFNN training on the UNSW-NB15 dataset, from the pre-correction pipeline (kept to show training dynamics only; see Table 5.2 for the corrected metrics). (a) Model trained with enriched data shows gradual convergence with a gap between training and validation loss. (b) Model trained on traffic-only data converges within a few epochs.

The model trained on the original dataset follows a different training path. Sub-figure 5.1(b) shows that both training and validation losses decrease during the first epochs. The values drop from about 0.6 and 0.47 to values close to zero. This change occurs before epoch ten. After this point the curves remain close to zero for the rest of the training process. The two curves remain almost identical. This behavior indicates that the model fits the dataset with little difficulty.

The contrast between the two learning curves reveals different learning conditions. The enriched dataset introduces information that describes user interaction with services. This additional structure changes the distribution of the training samples. The classifier must process both traffic patterns and behavioral signals. The learning process therefore requires more epochs to reach convergence. Because these curves come from the pre-correction pipeline, this contrast is reported as a description of training dynamics only; it is not taken as evidence that one model generalizes better than the other.

The rapid convergence observed in the model trained on the original dataset suggests a different learning mechanism. The network identifies discriminative patterns at an early stage of

training. The optimization process then reaches a stable state with near-zero loss. When a model reaches such values across both training and validation sets, the classifier may rely on patterns that exist only in the dataset structure. In network intrusion datasets this situation can occur when specific feature combinations provide direct separation between classes. In this pipeline, the corrected results below show that the near-zero loss came from data leakage.

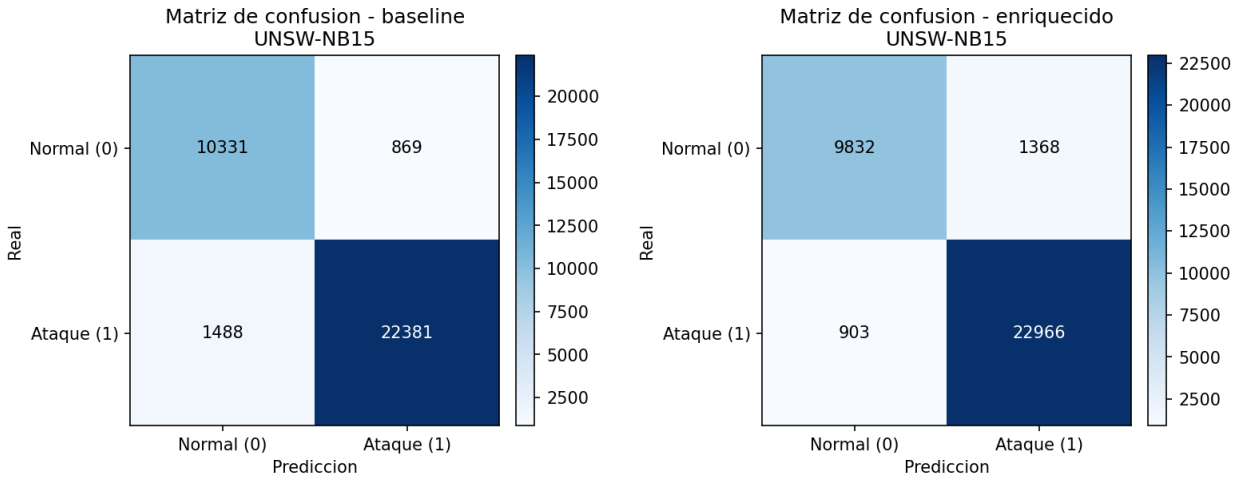


Fig. 5.2 Confusion matrices for the DFNN on UNSW-NB15, leakage-free pipeline. (a) Baseline: 869 false positives, 1,488 false negatives. (b) Enriched: 1,368 false positives, 903 false negatives. Color intensity represents the number of samples in each cell.

Within the pre-correction pipeline, Fig. 5.1 therefore illustrates how the enriched representation modifies the learning process. The model trained with enriched data faces a more structured input space. The model must build decision rules that integrate both traffic features and user behavior information. This condition increases the complexity of the learning task and produces a longer convergence process.

Fig. 5.2 presents the corrected confusion matrices for the two training settings, obtained after the label columns were removed from the input features and the split was moved ahead of every transformer. Each matrix reports true positives, true negatives, false positives, and false negatives on the same held-out test partition.

The baseline classifier, trained on traffic features alone, produces 869 false positives and 1,488 false negatives on a test set of 35,069 records, for an accuracy of 0.9328 and a ROC AUC of 0.9865. The enriched classifier produces 1,368 false positives and 903 false negatives on the same

test set, for an accuracy of 0.9353 and a ROC AUC of 0.9829. Neither model reaches a perfect score, and the small gap between them sits well inside the range the ablation study in Chapter 4 already attributed to added dimensionality rather than to behavioral signal.

Precision, recall, and specificity move in opposite directions between the two models. The baseline reaches a precision of 0.9626 and a recall of 0.9377, favoring fewer false alarms over catching every attack. The enriched model reaches a precision of 0.9438 and a recall of 0.9622, the reverse trade-off. Balanced accuracy lands at 0.9301 for the baseline and 0.9201 for the enriched model, computed directly from the confusion matrices above.

Fig. 5.3 shows the ROC curves for both models. The baseline reaches an AUC of 0.9865 against 0.9829 for the enriched model. A gap of this size, in the direction that favors the baseline, is consistent with what the ablation study already showed: applied across the whole dataset, behavioral enrichment does not add a signal that a traffic-only model lacks.

An earlier version of this pipeline reported 100% accuracy for the baseline model on this dataset. That number came from a leakage problem: the scaler, the class balancing step, and the autoencoders were fit on the full dataset before it was split into training, validation, and test partitions, so information from the test partition shaped every transformer applied to it. Once the split moves ahead of every transformer, as described in Chapter 1 and Chapter 4, the baseline accuracy drops to 0.9328, a figure that reflects the actual difficulty of separating attack traffic from normal traffic in this dataset.

Table 5.2 collects the corrected metrics for both models on UNSW-NB15.

TABLE 5.2

CORRECTED CLASSIFICATION METRICS ON UNSW-NB15 (LEAKAGE-FREE PIPELINE)

Metric	Baseline	Enriched
Accuracy	0.9328	0.9353
Precision	0.9626	0.9438
Recall	0.9377	0.9622
Specificity	0.9224	0.8779
Balanced accuracy	0.9301	0.9201
ROC AUC	0.9865	0.9829

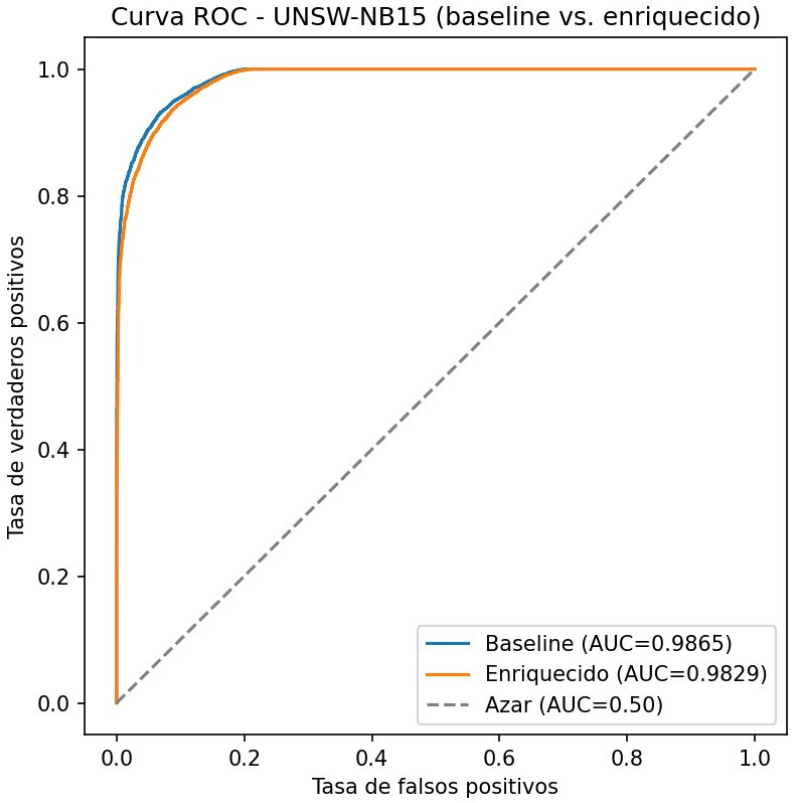


Fig. 5.3 ROC curves for the baseline and enriched classifiers on UNSW-NB15, computed on the leakage-free test partition.

No model in this comparison reaches 100% on any metric once the leakage is removed, and no claim in this chapter rests on a perfect score.

Beyond the single-split comparison above, the same baseline-versus-enriched comparison was repeated across three random seeds, with a paired t-test and Cohen's d computed for each metric. Table 5.3 reports the result.

TABLE 5.3

MULTI-SEED COMPARISON OF BASELINE AND ENRICHED CLASSIFIERS ON UNSW-NB15 (THREE SEEDS, PAIRED T-TEST)

Metric	Baseline (mean ± 95% CI)	Enriched (mean ± 95% CI)	p-value
Accuracy	0.9327 ± 0.0020	0.9311 ± 0.0040	0.561
Precision	0.9639 ± 0.0022	0.9475 ± 0.0036	0.008
Recall	0.9362 ± 0.0019	0.9516 ± 0.0103	0.077
F1-Score	0.9498 ± 0.0015	0.9495 ± 0.0033	0.873
ROC AUC	0.9869 ± 0.0007	0.9815 ± 0.0016	0.040
PR-AUC	0.9936 ± 0.0004	0.9904 ± 0.0010	0.041
MCC	0.8485 ± 0.0047	0.8414 ± 0.0083	0.284
Balanced accuracy	0.9307 ± 0.0026	0.9195 ± 0.0005	0.015
Specificity	0.9253 ± 0.0047	0.8875 ± 0.0094	0.010
Brier score	0.0435 ± 0.0006	0.0463 ± 0.0019	0.127

Five of the ten metrics, precision, ROC AUC, PR-AUC, balanced accuracy, and specificity, reach conventional significance ($p < 0.05$) in favor of the baseline; none reach significance in favor of the enriched model. This statistical check confirms, beyond the single split reported in Table 5.2, that applying behavioral enrichment across the whole UNSW-NB15 dataset does not help, and on several metrics measurably hurts, the classifier relative to the traffic-only baseline.

Cross-validation on the corrected pipeline retrains the full pipeline inside every fold: a new scaler, new autoencoders, a new adversarial alignment, and a new classifier, fit only on that fold's training data. An earlier version of this cross-validation reused a single scaler and a single pair of autoencoders across all five folds, which left a narrow form of leakage between folds even after the main train/test leakage was fixed.

Fig. 5.4 shows the fold-level distribution of accuracy, ROC AUC, and F1-score under this fully retrained cross-validation. The values sit close to the single-split results in Table 5.2 above, with the spread across folds visible directly in the boxplot rather than collapsed into a single mean.

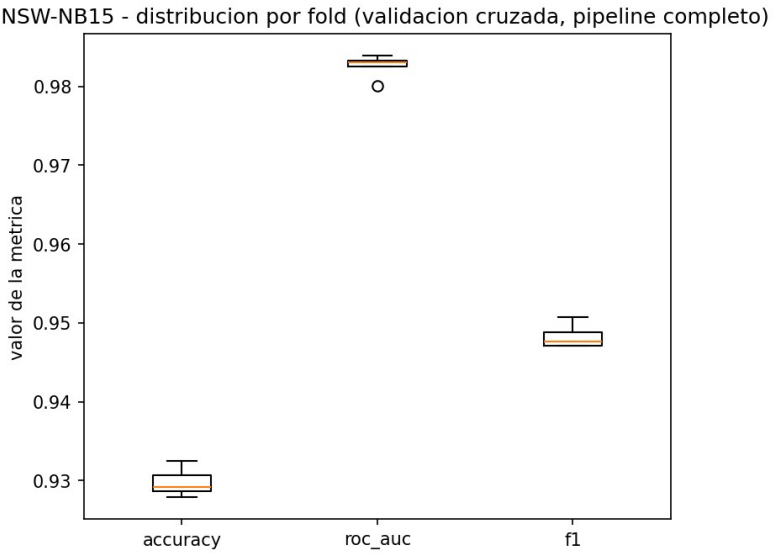


Fig. 5.4 Fold-level distribution of accuracy, ROC AUC, and F1-score for UNSW-NB15 under the fully retrained, leakage-free cross-validation.

Table 5.4 reports this cross-validation breakdown per fold in full, alongside the average across folds.

TABLE 5.4

CROSS-VALIDATION METRICS PER FOLD AND AVERAGE FOR THE DEEP FEEDFORWARD NEURAL NETWORK CLASSIFIER TRAINED ON THE ENRICHED UNSW-NB15 DATASET

Fold	Accuracy	Precision	Recall	F1-Score	ROC-AUC
1	0.9279	0.9461	0.9481	0.9471	0.9801
2	0.9325	0.9446	0.9570	0.9508	0.9830
3	0.9287	0.9564	0.9379	0.9471	0.9839
4	0.9307	0.9540	0.9437	0.9488	0.9833
5	0.9292	0.9541	0.9413	0.9476	0.9825
Average	**0.9298**	**0.9510**	**0.9456**	**0.9483**	**0.9826**

The cross-validation results are consistent with the single-split analysis: the enriched model keeps a stable level of performance across folds, with an average accuracy of 0.9298 and ROC AUC of 0.9826, in line with the single-split figures in Table 5.2 and, like them, not above the traffic-only baseline.

Stability across folds shows that the enriched pipeline is reproducible once every transformer is refit inside each fold. It does not show that the behavioral component adds discriminative information; that question is answered by the multi-seed comparison above and by the ablation study in Chapter 4.

Generalization beyond random partitions is a separate question, examined in Section 5.6.5.

5.6.2. Results on the CIC-IDS2017 Dataset

This section reports the classification results for the second dataset. Two model settings are compared. The first model uses the original network traffic data. The second model uses traffic data with behavioral enrichment. The enrichment step integrates user activity data with the network traffic representation. The goal is to study how this representation changes the learning

process and the classification results.

Fig. 5.5 shows the learning curves for both models. The curves present the training loss and the validation loss during the training epochs. Sub-figure 5.5(a) shows the model trained with enriched data. The training loss and validation loss decrease during the training process. Both curves move toward low loss values during the epochs. A gap between the curves remains during the process. This gap shows that the model does not fit the training data completely. The model continues to process variation in the dataset during training.

As with Fig. 5.1, the curves in Fig. 5.5 come from the pipeline as it stood before the leakage fix and are kept only to show convergence dynamics; the numbers below use the corrected pipeline.

Sub-figure 5.5(b) shows the model trained with the original dataset. The training loss and validation loss decrease in the first epochs. Both curves reach values near zero after a small number of epochs. The curves then remain close to zero for the rest of the training process. The two curves follow the same path. This pattern indicates that the classifier fits the dataset with little resistance during optimization.

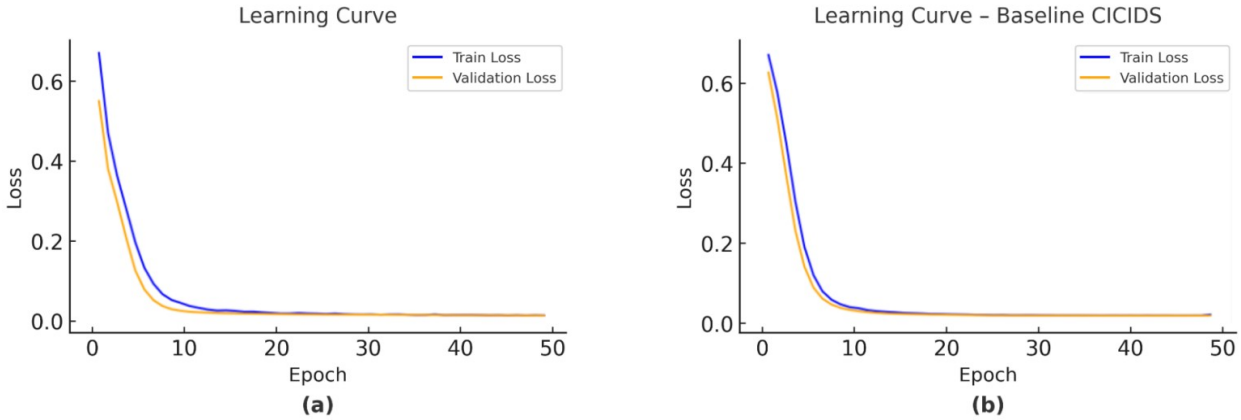


Fig. 5.5 Learning curves of the DFNN on the CIC-IDS2017 dataset, from the pre-correction pipeline (kept to show training dynamics only; see Table 5.5 for the corrected metrics). (a) Model trained with enriched data. (b) Model trained on traffic-only data.

The contrast between the two training processes shows two forms of learning. The enriched dataset includes both traffic patterns and user behavior patterns. The classifier must analyze these signals together. The learning process therefore requires more epochs before convergence. The gap between training and validation curves shows that the classifier continues to face variation in the

data space.

The model trained with the original dataset shows a different process. The classifier finds patterns that allow separation between classes during early epochs. The network reaches low loss values after a short training stage. This behavior suggests that the model captures patterns that belong to the dataset structure. When such patterns appear in both training and validation partitions, the classifier can reach loss values close to zero.

The training behavior observed in this dataset follows the same pattern found in the UNSW-NB15 dataset experiment. In both datasets the enriched model requires more epochs for convergence. In both cases the baseline model reaches low loss values in early epochs. This repeated pattern across datasets indicates that the enrichment process changes the learning conditions in a consistent way within the pre-correction pipeline.

Fig. 5.6 presents the corrected confusion matrices for CIC-IDS2017, obtained on the same held-out test partition after the leakage fix described in Section 5.6.1.

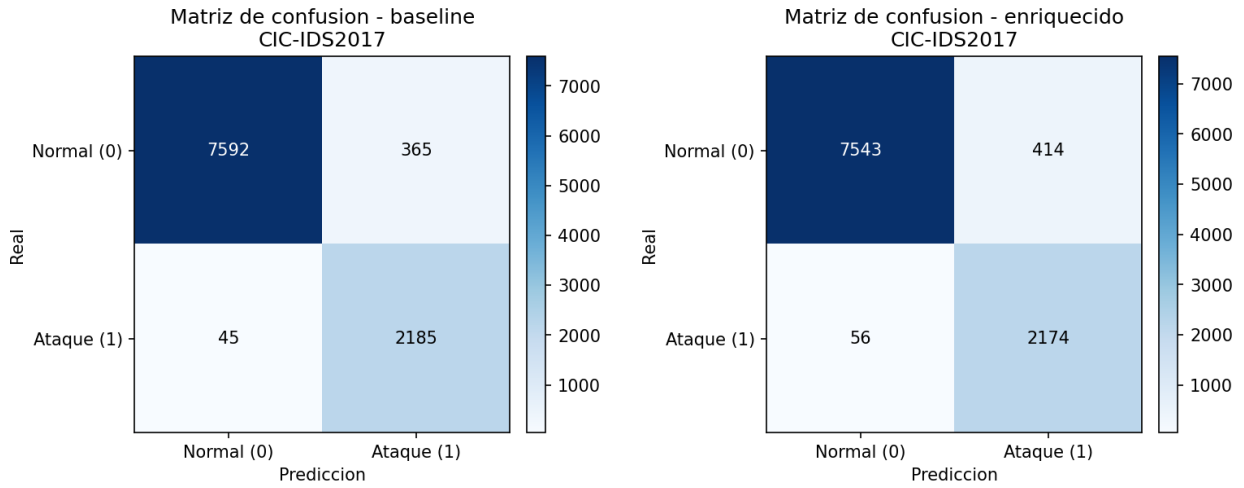


Fig. 5.6 Confusion matrices for the DFNN on CIC-IDS2017, leakage-free pipeline. (a) Baseline: 365 false positives, 45 false negatives. (b) Enriched: 414 false positives, 56 false negatives. Color intensity represents the number of samples in each cell.

The baseline classifier produces 365 false positives and 45 false negatives on a test set of 10,187 records, for an accuracy of 0.9598 and a ROC AUC of 0.9958. The enriched classifier produces 414 false positives and 56 false negatives, for an accuracy of 0.9539 and a ROC AUC of 0.9906. The baseline outperforms the enriched model on every metric here, the same direction

found in the ablation table in Chapter 4 for this dataset.

Both models make errors on samples whose traffic features overlap across classes. Adding the behavioral embedding does not resolve those cases and slightly increases their number: 470 errors for the enriched model against 410 for the baseline.

Neither model reaches a perfect score once the leakage is removed. The baseline still produces fewer errors than the enriched model on this dataset, but 365 false positives and 45 false negatives are far from the zero-error result the pre-correction pipeline had reported; that earlier result reflected the scaler, class balancer, and autoencoders having been fit on the full dataset before the split, not genuine separability of the classes.

Table 5.5 collects the corrected metrics for both models on CIC-IDS2017. Precision reaches 0.8569 for the baseline against 0.8400 for the enriched model; recall reaches 0.9798 against 0.9749; balanced accuracy reaches 0.9670 against 0.9615. Every one of these figures comes directly from the confusion matrices above. Fig. 5.7 shows the corresponding ROC curves: the baseline curve sits above the enriched curve across the full range of false-positive rates, consistent with its higher AUC (0.9958 against 0.9906).

TABLE 5.5

CORRECTED CLASSIFICATION METRICS ON CIC-IDS2017 (LEAKAGE-FREE PIPELINE)

Metric	Baseline	Enriched
Accuracy	0.9598	0.9539
Precision	0.8569	0.8400
Recall	0.9798	0.9749
Specificity	0.9541	0.9480
Balanced accuracy	0.9670	0.9615
ROC AUC	0.9958	0.9906

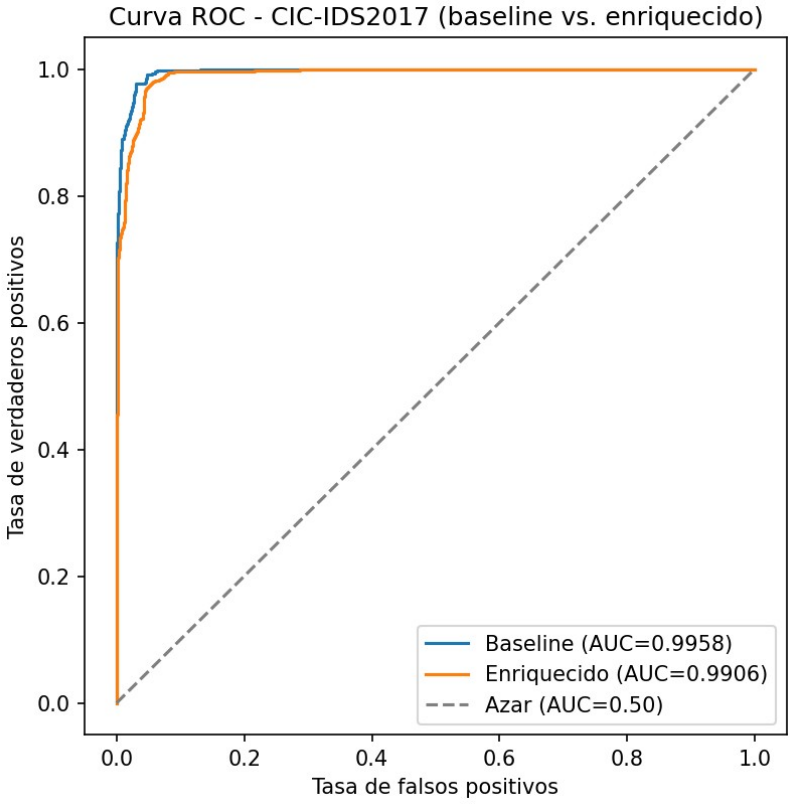


Fig. 5.7 ROC curves for the baseline and enriched classifiers on CIC-IDS2017, computed on the leakage-free test partition.

As with UNSW-NB15, no model here reaches 100% on any metric once the leakage is removed.

As with UNSW-NB15, the baseline-versus-enriched comparison was repeated across three random seeds on CIC-IDS2017. Table 5.6 reports the result.

TABLE 5.6

MULTI-SEED COMPARISON OF BASELINE AND ENRICHED CLASSIFIERS ON CIC-IDS2017 (THREE SEEDS, PAIRED T-TEST)

Metric	Baseline (mean ± 95% CI)	Enriched (mean ± 95% CI)	p-value
Accuracy	0.9811 ± 0.0209	0.9549 ± 0.0075	0.138
Precision	0.9359 ± 0.0776	0.8453 ± 0.0281	0.154
Recall	0.9846 ± 0.0059	0.9728 ± 0.0045	0.151
F1-Score	0.9589 ± 0.0437	0.9044 ± 0.0141	0.137
ROC AUC	0.9974 ± 0.0017	0.9914 ± 0.0019	0.011
PR-AUC	0.9925 ± 0.0065	0.9737 ± 0.0059	0.016
MCC	0.9478 ± 0.0551	0.8789 ± 0.0172	0.137
Balanced accuracy	0.9823 ± 0.0151	0.9613 ± 0.0033	0.115
Specificity	0.9801 ± 0.0254	0.9499 ± 0.0108	0.155
Brier score	0.0136 ± 0.0117	0.0324 ± 0.0049	0.056

ROC AUC and PR-AUC reach conventional significance ($p < 0.05$) in favor of the baseline; the remaining metrics move in the same direction without reaching significance at this seed count, consistent with the wider confidence intervals CIC-IDS2017 produces relative to UNSW-NB15. No metric favors the enriched model.

As with UNSW-NB15, cross-validation on the corrected pipeline retrains the scaler, the autoencoders, the adversarial alignment, and the classifier inside every fold, using only that fold's training data.

Table 5.7 reports this same cross-validation breakdown per fold for CIC-IDS2017.

TABLE 5.7

CROSS-VALIDATION METRICS PER FOLD AND AVERAGE FOR THE DEEP FEEDFORWARD NEURAL NETWORK CLASSIFIER TRAINED ON THE ENRICHED CIC-IDS2017 DATASET

Fold	Accuracy	Precision	Recall	F1-Score	ROC-AUC
1	0.9402	0.7922	0.9852	0.8782	0.9919
2	0.9614	0.8670	0.9731	0.9170	0.9941
3	0.9666	0.8848	0.9744	0.9274	0.9962
4	0.9675	0.8864	0.9767	0.9294	0.9959
5	0.9445	0.8084	0.9785	0.8853	0.9925
Average	**0.9561**	**0.8478**	**0.9776**	**0.9075**	**0.9941**

Fig. 5.8 shows the fold-level distribution of accuracy, ROC AUC, and F1-score under this fully retrained cross-validation, at the same order of magnitude as the single-split results in Table 5.5 above.

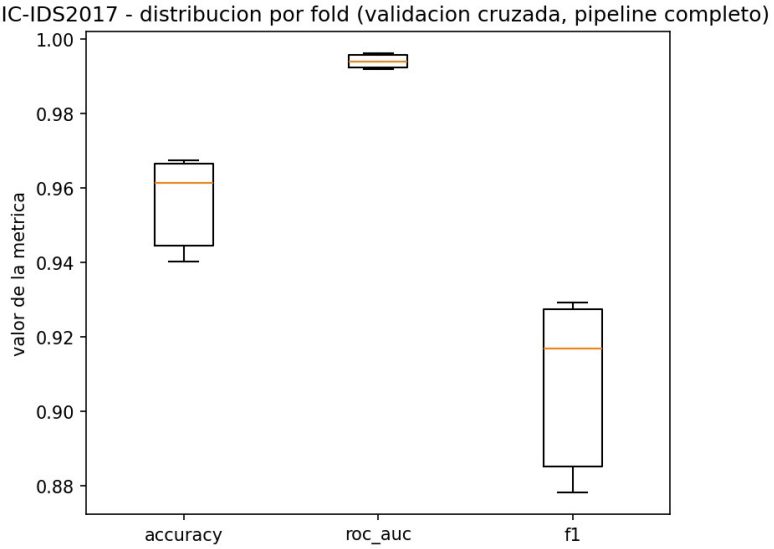


Fig. 5.8 Fold-level distribution of accuracy, ROC AUC, and F1-score for CIC-IDS2017 under the fully retrained, leakage-free cross-validation.

Both datasets follow the same direction once the pipeline is corrected: the baseline model

outperforms the behaviorally enriched model on nearly every metric, and the gap in both cases falls inside the range the ablation controls in Chapter 4 already produced with no behavioral content at all.

This chapter does not stop at this negative result. Section 5.6.4 applies the same behavioral embedding through a different combination rule, a confidence cascade that reaches only the traffic records the base classifier cannot yet resolve, and finds a result that survives replication on UNSW-NB15.

Applied across the whole dataset, in both UNSW-NB15 and CIC-IDS2017, behavioral enrichment does not raise classifier performance over a traffic-only baseline free of data leakage.

5.6.3. Reference Baseline Classifiers

A deep feedforward network is only one architecture, and a gap between a baseline and an enriched version of it could in principle come from that one architecture rather than from the enrichment itself. Logistic regression, Random Forest, XGBoost, SVM, and an MLP built with scikit-learn were trained on the same splits and the same features, baseline and enriched, to check whether the pattern above is specific to the neural network.

It is not. On UNSW-NB15, XGBoost trained on traffic features alone reaches an accuracy of 0.9515 and a ROC AUC of 0.9933; the enriched version reaches 0.9323 and 0.9868, below the baseline on both metrics. On CIC-IDS2017, Random Forest trained on traffic features alone reaches 0.9972; the enriched version reaches 0.9861. Logistic regression, SVM, and the MLP followed the same direction across both datasets. The gap between baseline and enriched performance shown throughout this chapter is not an artifact of the deep feedforward architecture.

5.6.4. Confidence Cascade Enrichment

Every result so far applies enrichment to the entire dataset, and every one of them comes out flat against a synthetic control. A different rule was tested next: let the traffic-only classifier resolve every record it can resolve with confidence, and reserve the behavioral embedding for the narrow band of records where that classifier's output sits close to 0.5, its decision boundary. Three bandwidths around 0.5 were fixed before any result was seen: a narrow band from 0.40 to 0.60, a medium band from 0.30 to 0.70, and a wide band from 0.20 to 0.80.

An exploratory phase ran this cascade with six seeds, computing the mean difference between the real behavioral embedding and a noise control, a 95% confidence interval, a paired t-test, and a bootstrap of the difference over 10,000 resamples. A confirmatory phase then ran the same cascade once more, on a seed held out of the exploratory phase, to check that the result did not depend on having tried several seeds and kept the best one.

On UNSW-NB15, the real embedding beat the noise control in ROC AUC across all three bands, in the exploratory phase and again in the confirmatory run. The gap grows with the band: 0.022 in the narrow band, widening to 0.051 in the wide band, with a large effect size in every case. Statistical significance moves in the same direction: p equals 0.030 in the narrow band, 0.0007 in the medium band, and 0.0003 in the wide band. Fig. 5.9 shows the exploratory confidence interval for the medium band, and Fig. 5.10 shows the underlying seed-level spread of ROC AUC for the same band.

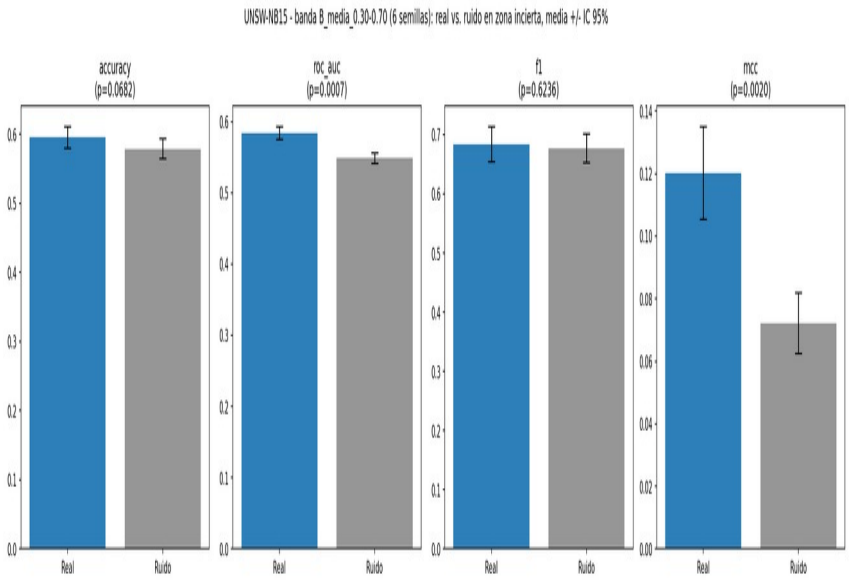


Fig. 5.9 Exploratory 95% confidence interval for the ROC AUC difference between the real behavioral embedding and a noise control, confidence-cascade enrichment, medium band, UNSW-NB15, across six seeds.

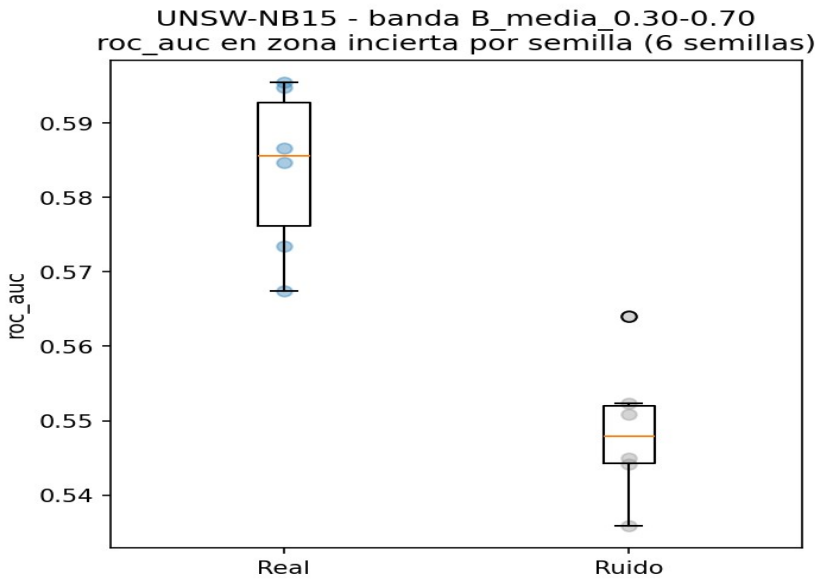


Fig. 5.10 Seed-level ROC AUC for the real embedding against its noise control, confidence-cascade enrichment, medium band, UNSW-NB15.

CIC-IDS2017 does not repeat this result, for a reason tied to dataset size rather than to the mechanism itself: the uncertainty band around 0.5 captures only 32 to 170 records here, depending on the band width, too few for the comparison to hold together across seeds. The evidence on this dataset is reported as inconclusive rather than forced into either direction.

Taken together with the ablation results earlier in this chapter, the confidence cascade is the one combination rule, of four tested, for which behavioral enrichment shows a gain that replicates on a held-out seed. That gain is specific to UNSW-NB15 and to the narrow band of traffic records a classifier cannot yet resolve on its own; it is not a general property of adding behavioral information to network traffic.

5.6.5. Generalization Across Partitions

Every result above splits data at random, so two records from the same capture file or the same time window can land on opposite sides of the train/test boundary. CIC-IDS2017 allows a stricter check: splitting by the source capture file, training on one subset of files and testing on files the model never saw during training.

That split is not encouraging, and it is reported as it stands. A model trained this way detects none of the attacks present in the unseen capture files, a recall of zero. UNSW-NB15 does not

allow the equivalent check: the 42-feature processed version of this dataset used throughout this thesis does not retain the original timestamp column, so a temporal split could not be built. Both limits are stated here rather than left implicit.

5.7. Summary of Deep Learning for Network Anomaly Detection

This chapter studied deep learning models for network anomaly detection. The study used traffic data and behavior data. Two input settings were tested. The first used traffic features from the datasets. The second used a joint vector that combined traffic features with behavior embeddings. The goal was to study how behavior context affects model learning and detection results.

The model used in this study was a deep feedforward neural network. The network received feature vectors that describe network flows. In the enriched setting the vector also included behavior signals from user activity logs. The model learned a mapping between the input vector and the traffic label. Training used labeled data and binary cross entropy loss. Model performance was measured with accuracy, precision, recall, F1-score, ROC AUC, PR-AUC, the Matthews correlation coefficient, balanced accuracy, specificity, and the Brier score.

Experiments used two benchmark datasets: the UNSW-NB15 dataset and the CIC-IDS2017 dataset. In each dataset two models were trained. The baseline model used traffic features only. The second model used the enriched representation that combined traffic and behavior embeddings.

The loss curves from the pre-correction pipeline showed two learning patterns, but the corrected pipeline traces that contrast to a specific cause rather than to a general property of enrichment: an earlier version of the code fit the scaler, the class balancer, and the autoencoders on the full dataset before splitting it, which let information from the test partition leak into every transformer applied to it.

Once the split moves ahead of every transformer, the baseline no longer reaches a perfect score on either dataset, and the enriched model does not outperform it. Figs. 5.2 and 5.6 report both confusion matrices, and Tables 5.2 and 5.5 the corresponding metrics.

Applied across the full dataset, behavioral enrichment does not raise accuracy, ROC AUC, or any other metric reported in this chapter, on either dataset, and five classical baselines confirm that this is not an artifact of the neural network architecture.

The one setting in which behavioral enrichment shows a gain that survives a confirmatory run is the confidence cascade described in Section 5.6.4, restricted to UNSW-NB15 and to the narrow band of traffic records the base classifier cannot yet resolve.

This chapter therefore supports a bounded version of the hypothesis stated in Chapter 1: behavioral enrichment does not improve traffic-only detection in general, but under the confidence cascade it produces a gain in UNSW-NB15 that holds across seeds and a held-out confirmatory run. Chapter 6 discusses what this bounded result means for the framework as a whole.

Chapter 6 presents a broader discussion of the results reported in the previous chapters. It revisits the findings from the experiments with the UNSW-NB15 dataset and the CIC-IDS2017 dataset. The chapter examines the behavior of the baseline model and the model trained with enriched data. It analyzes the training curves, the confusion matrices, and the classification metrics. The goal is to interpret the results in relation to model learning, feature representation, and anomaly detection in network traffic.

5.8. Published Work

The related work was published as:

Najar Juárez, C. S. (2024). *Towards a user network profiling for internal security using top-k ranking similarity measures using machine learning*. In Encuentro Nacional *de Computación, Eureka ENC 2024*, Cd. Madero, Tamaulipas, México.

Najar Juárez, C. S., & Domínguez Pérez, L. J. (2025). *Cross-domain adversarial alignment for network anomaly detection through behavioral embedding enrichment*. *Computers, 14*(11), 450. https://doi.org/10.3390/computers14110450

6. Discussion

6.1. Integration of Behavioral and Network Traffic Information

The first part of this study focuses on the construction of user behavior profiles and their relation with network traffic analysis. The results show that user activity contains patterns that can support the analysis of network events. These patterns appear in browsing actions, session structure, and interaction sequences recorded during web activity monitoring. When these patterns are processed through representation learning, each user can be expressed as a vector that summarizes the structure of the observed behavior. These vectors form the basis for behavioral embeddings used in the later stages of the study.

Behavioral embeddings allow the transformation of complex activity logs into numerical representations. Each vector encodes relations between actions that occur during user sessions. These relations include the relative frequency of requests to each service, the dispersion of activity measured by entropy, and the principal components of the interaction profile. When the dimensionality of the data is reduced through an autoencoder, the resulting latent vectors capture the structure of the activity in a compact form. These representations preserve information that describes how users interact with network services [Benova-Hudec-24; He-Yin-21].

After the generation of embeddings, clustering methods identify groups of users with similar activity patterns. Each cluster represents a behavioral profile. A profile is not tied to a specific individual. Instead, it represents a pattern of interaction observed in the monitored data. Some profiles correspond to intensive browsing behavior, while others represent periodic or limited activity. The clustering process organizes the behavioral space and produces a set of profiles that summarize the variability of user actions across sessions [Ikotun-Ezugwu-23; Singh-Girdhar-22].

The presence of these profiles introduces an additional layer of information that is not present in network traffic features alone. Traditional anomaly detection methods rely on packet features such as protocol type, packet size, time intervals, and port numbers. These features

describe the structure of the communication between devices. However, they do not capture the behavioral context in which the communication occurs. Network packets reveal how data moves across the network, but they do not describe why the activity occurs or which behavioral pattern it follows [Wang-Jones-20; Bashurov-Safonov-23].

The integration of behavioral profiles with network traffic addresses this limitation. In the proposed framework, each network traffic record is paired with a behavioral embedding derived from user activity. This association is based on similarity between latent representations. The similarity measure selects the behavioral vector that is closest to the traffic representation in the latent space. The selected vector is concatenated with the traffic features to form a unified representation for each record [Modell-Larson-21; Pandhurnekar-Iyyappan-23].

This process produces a feature vector that contains two types of information. The first part describes the technical characteristics of the network communication. The second part represents the behavioral context associated with the activity. The combination of both domains creates a representation that links user actions with network events. Each data instance becomes a joint description of traffic behavior and user interaction patterns [Alshehri-Khan-23; P-Palaniappan-24].

The conceptual value of this integration lies in the complementary nature of the two domains. Traffic features describe the physical or protocol-level aspects of communication. Behavioral embeddings describe patterns of user activity that produce the communication. When these domains are analyzed together, the model receives information about both the mechanism of the network exchange and the context in which it occurs [Shafi-Lashkari-24].

This integration also expands the representation space used by the anomaly detection model. Instead of learning patterns from network traffic alone, the model learns relations between traffic structure and behavioral context. The enriched representation increases the dimensionality of the feature space and introduces new relations between variables. In the proposed system, the enriched vectors combine a traffic representation with a behavioral profile to produce a joint feature vector used for classification [Singh-Jindal-23].

Another important aspect of this integration is the preservation of domain structure. Behavioral embeddings and traffic embeddings originate from different datasets and describe

different phenomena. Direct merging of these datasets could distort the structure of the data. To address this issue, the system applies adversarial alignment to map both domains into a shared latent space before the final integration step. This alignment ensures that the two types of embeddings can coexist within the same representation without losing the internal structure of each domain [Xi-Zhou-22].

From a conceptual perspective, this process can be interpreted as a form of contextual augmentation. Behavioral profiles provide information about user interaction patterns that surround network events. When this information is attached to traffic data, each record gains an additional layer of interpretation. The anomaly detection model can then evaluate not only whether a traffic pattern deviates from expected packet structures, but also whether it deviates from expected behavioral patterns associated with the user activity [Tun-Nyaung-20; Duan-Fu-23].

The enrichment stage keeps the two domains distinguishable in the latent space, as Fig. 4.1 shows, and the ablation study in Chapter 4 tested directly whether that separation carries information a classifier can use. It mostly does not: replacing the real behavioral embedding with zeros, noise, or a random assignment reproduces the same classification outcome as the real embedding, in both datasets, once enrichment reaches the full traffic set.

The rest of this chapter reads that negative result together with the one positive result Chapter 5 did find: the confidence cascade, which restricts enrichment to traffic records a classifier cannot yet resolve and produces a gain that holds up under a confirmatory run on UNSW-NB15.

6.2. Impact of Behavioral Enrichment on Anomaly Detection

The second part of this discussion turns to what Chapter 5 measured directly: the effect of behavioral enrichment on classifier performance, once the leakage found in an earlier version of the pipeline is removed.

With the leakage removed, the baseline classifier reaches an accuracy of 0.9328 on UNSW-NB15 and 0.9598 on CIC-IDS2017. Neither figure is perfect, and neither invites a memorization story; they read as the ordinary difficulty of the task. Against that baseline, a behaviorally enriched classifier trained on the whole dataset does not gain ground, on either dataset, under the deep

feedforward network or under five classical models tested for comparison. Ablation controls with no behavioral content at all, a zero vector, Gaussian noise, a random assignment, land in the same range. Chapter 5 traces one condition under which the pattern changes: a confidence cascade that reserves the behavioral embedding for traffic records the base classifier cannot yet resolve. There, on UNSW-NB15, the real embedding beats a noise control across three band widths, with the gap growing from 0.022 to 0.051 in ROC AUC and holding on a seed never used during exploration. CIC-IDS2017 could not confirm or rule out the same effect, since its band of uncertain cases held too few records to compare.

6.3. Implications and Limitations

A few implications follow from the results above, and they point in a more modest direction than the framework first suggested.

The adversarial alignment mechanism does combine heterogeneous data sources, behavioral logs and network flows, into a shared representation without collapsing either domain's structure, and the semantic check in Chapter 4 confirms that the resulting match is not equivalent to a random pairing. What that alignment does not do, on its own, is raise detection accuracy. The value of cross-domain alignment in this thesis turns out to sit in the confidence cascade built on top of it, not in the alignment step by itself.

Despite these contributions, several limitations remain. The behavioral dataset used in the study represents a specific collection of monitored web activity sessions. User behavior may vary across organizations, geographic regions, and network environments. Different populations of users may produce behavioral patterns that differ from those observed in this dataset [He-Yin-21; Alshehri-Khan-23].

Two further limitations weigh more heavily than the others. Recall drops to zero when CIC-IDS2017 is split by capture file instead of at random, so a model trained on this pipeline does not yet generalize to traffic captured at a different time. UNSW-NB15 could not be checked the same way, because the processed 42-feature version used throughout this thesis drops the original timestamp column.

Another limitation concerns scalability. The proposed framework includes several stages, including behavioral profiling, adversarial alignment, and classification. Each stage introduces computational cost. The evaluation in this study focuses on experimental datasets and offline analysis. Additional work is required to determine whether the full pipeline can operate in real network environments with large traffic volumes [Tun-Nyaung-20; P-Palaniappan-24].

The evaluation also focuses on two benchmark datasets. While these datasets represent common references in intrusion detection research, they cannot capture the diversity of network infrastructures and attack strategies observed in operational environments. Future research should evaluate the enrichment approach across additional datasets and network contexts [Kasongo-Sun-20; Abiramasundari-Ramaswamy-25].

Further work may also explore alternative strategies for representation learning and feature integration. Other dimensionality reduction techniques could replace the autoencoder used for behavioral embeddings. Additional data sources, such as device fingerprints or application logs, could extend the representation space beyond user activity and packet features [Modell-Larson-21; Shafi-Lashkari-24].

Finally, the interpretability of enriched models remains an open question. Deep learning classifiers operate through complex transformations of high-dimensional feature vectors. Understanding how behavioral context influences specific anomaly decisions requires methods that can explain model predictions. Research on explainable machine learning and graph-based detection may help address this challenge [Marfo-Tosh-24; Pandhurnekar-Iyyappan-23].

Read together, the chapters above support a narrower claim than the one this thesis set out with. Behavioral enrichment, applied across a full traffic dataset, does not raise anomaly detection performance over a leakage-free baseline. Restricted to the cases a traffic classifier cannot yet resolve, it does, on UNSW-NB15, under a confirmatory test. The contribution of this work sits in the experimental frame that tells the two apart, the ablation controls and the confidence cascade, more than in the behavioral dataset or the alignment mechanism on their own.

7. Conclusions and Future Work

7.1. Conclusions

This thesis addressed the general objective of developing a method for anomaly detection in network traffic through the integration of user behavioral profiling and deep learning. The results obtained across each of the specific objectives are summarized below.

Specific Objective 1, to collect *web activity data and represent user interaction through feature vectors and latent representations*, was addressed through proxy-monitored web sessions and generated activity records totaling 150 samples across four groups: network analysts, programmers, mechatronics students, and a fourth group of records labeled as atypical (anomalous) user behavior. Interaction logs were transformed into behavior vectors that included normalized service frequencies, entropy values, and principal component features. An autoencoder model then produced latent representations of these vectors, compressing behavioral patterns into a lower-dimensional space that captured structure not visible in the original features.

Specific Objective 2, to align *behavioral embeddings and network traffic features in a shared representation space*, was addressed through adversarial autoencoders trained to map behavioral and traffic features into a common latent space. A discriminator network guided the alignment process by distinguishing between embeddings from both domains while the autoencoders worked to produce indistinguishable representations. For each traffic instance, the most similar behavioral embedding was identified using cosine similarity and concatenated with the traffic embedding to produce an enriched joint representation. Clustering quality was confirmed through silhouette scores and Davies–Bouldin indices, which showed that latent representations improved cluster separation relative to the original feature space.

Specific Objective 3, to design and train deep learning models that use enriched representations for anomaly detection, was addressed through a deep feedforward network evaluated on UNSW-NB15 and CIC-IDS2017 with a leakage-free pipeline. The network trained

on enriched features reached an accuracy of 0.9353 and a ROC AUC of 0.9829 on UNSW-NB15, and an accuracy of 0.9539 and a ROC AUC of 0.9906 on CIC-IDS2017. Five classical models, logistic regression, Random Forest, XGBoost, SVM, and an MLP, were trained under the same conditions to check that this architecture was not the reason behind any gap found in Objective 4.

Specific Objective 4, to compare models trained on traffic data against models trained on enriched data under common evaluation metrics, was addressed through a baseline classifier trained under identical splits, architecture, epochs, loss function, and seeds as the enriched classifier. The baseline reached 0.9328 accuracy on UNSW-NB15 and 0.9598 on CIC-IDS2017, figures that reflect the task's real difficulty rather than the perfect scores an earlier, leaking version of the pipeline had reported. Applied across the whole dataset, the enriched classifier did not outperform this baseline on either dataset, a result an ablation study with six synthetic controls and five classical models confirmed was not an artifact of one architecture or one dataset split. A confidence cascade that restricts enrichment to traffic records near the base classifier's decision boundary did outperform the baseline, on UNSW-NB15, with the gain replicated on a confirmatory seed.

Taken as a whole, this thesis reaches a narrower conclusion than the one it set out to test. Behavioral profiling and network traffic analysis can be combined through representation learning, and the resulting alignment is not mathematically arbitrary, as the semantic validity check in Chapter 4 shows. That combination does not, on its own, raise anomaly detection performance above a leakage-free traffic-only baseline. It does raise performance under one specific condition, a confidence cascade applied to the traffic records a base classifier cannot yet resolve, and only on UNSW-NB15. The general objective, stated at the start of this thesis as developing a method that integrates behavioral profiling into anomaly detection, is met in that bounded form: not as a representation that always helps, but as an experimental frame, ablation controls plus a confidence cascade, that tells the two cases apart with evidence rather than assumption.

7.2. Future Work

Future work may extend the proposed framework through the integration of additional sources of behavioral and network data, allowing the construction of richer representations of activity

patterns. Further research may also explore advanced representation learning methods and sequence-based models to capture temporal dependencies within traffic and user interaction, as these architectures require substantially greater computational resources than those available during this study.

Another direction involves the incorporation of models that describe relationships between entities in network environments, as well as approaches that provide interpretability of detection decisions. These extensions, together with the training of models on larger and more heterogeneous datasets, can support the development of more robust anomaly detection systems based on integrated representations of network activity.

Three directions follow directly from the results in this thesis rather than from general extensions of the framework. The confidence cascade needs testing on datasets beyond UNSW-NB15 to check whether its gain generalizes past the one dataset where the current evidence holds. CIC-IDS2017 needs a larger uncertainty band, through more data or a different confidence threshold, before the cascade can be confirmed or ruled out there. The recall-zero result under a by-file split on CIC-IDS2017 also needs its own investigation, separate from the enrichment question, since it points to a generalization gap in the traffic classifier itself.

This page intentionally left blank.

Appendix

A. LIST OF PUBLICATIONS

A.1. JOURNAL PAPERS AND CONFERENCE PROCEEDINGS

1) Najar Juárez, C. S., & Domínguez Pérez, L. J. (2025). *Cross-domain adversarial alignment for network anomaly detection through behavioral embedding enrichment*. Computers, 14(11), 450. https://doi.org/10.3390/computers14110450
2) Najar Juárez, C. S. (2024). *Towards a user network profiling for internal security top-k rankings similarity measures using K-means. In* J. Mejía, M. Muñoz, A. Rocha, Y. Hernández Pérez, & H. Avila-George (Eds.), *New perspectives in software engineering* (Studies in Computational Intelligence, Vol. 1135). Springer. https://doi.org/10.1007/978-3-031-50590-4_19
3) Najar Juárez, C. S. (2023). *Towards a user network profiling for internal security top-k rankings similarity measures using K-means. In International Conference on Software Process Improvement (CIMPS 2023)*, Cuernavaca, Morelos, México.
4) Najar Juárez, C. S. (2024). *Towards a user network profiling for internal security using top-k ranking similarity measures using machine learning*. In Encuentro Nacional *de Computación, Eureka ENC 2024*, Cd. Madero, Tamaulipas, México.

B. Source Code of the Project

1. Data Preparation

1.1 Google Drive Mount

```
from google.colab import drive
drive.mount('/content/drive')
```

1.2 Charles Proxy — Feature Engineering

Dependencies and Data Loading

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from scipy.stats import entropy

main_data = pd.read_csv('/content/drive/MyDrive/Data/BD_Final_5.3.csv')
data = main_data.drop(columns=['Usuario', 'IP Address', 'categoria']).copy()
```

Normalization, Shannon Entropy, and PCA

```
# Interaction frequency normalization
data['total_interacciones'] = data.sum(axis=1)
df_normalizado =
data.drop(columns=['total_interacciones']).div(data['total_interacciones'], axis=0)

# Shannon entropy per user
def calcular_entropia(interacciones):
    total_interacciones = np.sum(interacciones)
    if total_interacciones == 0:
        return 0
    probabilidades = interacciones / total_interacciones
    entropia = -np.sum(probabilidades * np.log(probabilidades + 1e-9))
    return entropia

df_normalizado['entropia_actividad'] = data.apply(lambda row:
calcular_entropia(row), axis=1)

# PCA — two principal components appended to feature set
pca = PCA(n_components=2)
componentes_principales =
pca.fit_transform(df_normalizado.drop(columns=['entropia_actividad']))
df_normalizado['PCA_1'] = componentes_principales[:, 0]
df_normalizado['PCA_2'] = componentes_principales[:, 1]
```

1.3 Behavioral Clustering

Autoencoder and Agglomerative Clustering

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
```

```
from keras.models import Model
from keras.layers import Input, Dense
from keras.optimizers import Adam
from keras.regularizers import l2

data_scaled = df_normalizado.copy()
input_dim = data_scaled.shape[1]

# Autoencoder architecture
input_layer = Input(shape=(input_dim,))
encoding_dim = 2
encoded = Dense(encoding_dim, activation='relu', kernel_regularizer=l2(0.01))
(input_layer)
decoded = Dense(input_dim, activation='sigmoid')(encoded)

autoencoder = Model(inputs=input_layer, outputs=decoded)
encoder = Model(inputs=input_layer, outputs=encoded)

autoencoder.compile(optimizer=Adam(), loss='mean_squared_error')
autoencoder.fit(data_scaled, data_scaled, epochs=300, batch_size=128, shuffle=True,
                validation_data=(data_scaled, data_scaled))

encoded_data = encoder.predict(data_scaled)

# Silhouette score for cluster range 2-10
silhouette_scores_agglomerative_hybrid = []
for n_clusters in range(2, 11):
    agglomerative = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')
    agglomerative_labels = agglomerative.fit_predict(encoded_data)
    score = silhouette_score(encoded_data, agglomerative_labels)
    silhouette_scores_agglomerative_hybrid.append(score)

optimal_clusters_agglomerative = range(2, 11)[

silhouette_scores_agglomerative_hybrid.index(max(silhouette_scores_agglomerative_hy
brid))
]
```

1.4 Clustering Visualization and Metrics

```
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, calinski_harabasz_score,
davies_bouldin_score, pairwise_distances

data_scaled = df_normalizado.drop(columns=['cluster'], errors='ignore').copy()

silhouette_scores_kmeans = []
silhouette_scores_agglomerative = []

for n_clusters in range(2, 11):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(data_scaled)
    silhouette_scores_kmeans.append(silhouette_score(data_scaled, kmeans.labels_))

    agglomerative = AgglomerativeClustering(n_clusters=n_clusters)
    agglomerative.fit(data_scaled)
    silhouette_scores_agglomerative.append(silhouette_score(data_scaled,
agglomerative.labels_))

# Optimal cluster counts
optimal_clusters_kmeans = range(2, 11)
[silhouette_scores_kmeans.index(max(silhouette_scores_kmeans))]
```

```
optimal_clusters_agglomerative = range(2, 11)
[silhouette_scores_agglomerative.index(max(silhouette_scores_agglomerative))]
optimal_clusters_agglomerative_hybrid = range(2, 11)[

silhouette_scores_agglomerative_hybrid.index(max(silhouette_scores_agglomerative_hy
brid))
]

# Advanced clustering metrics (2-cluster configuration)
agglomerative = AgglomerativeClustering(n_clusters=2)
df_normalizado['cluster'] = agglomerative.fit_predict(encoded_data)

silhouette_avg       = silhouette_score(encoded_data, df_normalizado['cluster'])
calinski_harabasz_avg = calinski_harabasz_score(encoded_data,
df_normalizado['cluster'])
davies_bouldin_avg   = davies_bouldin_score(encoded_data,
df_normalizado['cluster'])

def dunn_index(X, labels):
    intra_cluster_distances = []
    inter_cluster_distances = []
    for cluster in np.unique(labels):
        cluster_points = X[labels == cluster]
        intra_cluster_distances.append(np.max(pairwise_distances(cluster_points)))
        for other_cluster in np.unique(labels):
            if cluster != other_cluster:
                other_cluster_points = X[labels == other_cluster]

inter_cluster_distances.append(np.min(pairwise_distances(cluster_points,
other_cluster_points)))
    return np.min(inter_cluster_distances) / np.max(intra_cluster_distances)

dunn_index_avg = dunn_index(encoded_data, df_normalizado['cluster'])
```

Final Behavioral Profile Dataset

```
agglomerative = AgglomerativeClustering(n_clusters=2)
df_normalizado['cluster'] = agglomerative.fit_predict(encoded_data)
charles_data = df_normalizado.copy()
```

1.5 UNSW-NB15 — Data Loading and Preprocessing

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

unswnb15_data = pd.read_csv(
    '/content/drive/MyDrive/0. B/2024/MLE/Proyectos Activos 2024/Cristian -
Profiling/'
    'Data/CSV Files/Training and Testing Sets/UNSW_NB15_training-set.csv'
)

# Select numeric columns, excluding identifiers
columns_to_exclude = ['srcip', 'dstip', 'proto', 'state', 'service', 'Stime',
'Ltime', 'attack_cat']
unsw_numeric = unswnb15_data.drop(columns=columns_to_exclude, errors='ignore')
```

2. UNSW-NB15 — Cross-Domain Behavioral Enrichment

2.1 Adversarial Autoencoder Training

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import cosine_similarity

# Preprocessing
charles_numeric = charles_data.select_dtypes(include=['int64', 'float64'])
scaler_unsw    = StandardScaler()
unsw_scaled    = scaler_unsw.fit_transform(unsw_numeric)
scaler_charles = StandardScaler()
charles_scaled = scaler_charles.fit_transform(charles_numeric)

unsw_tensor    = torch.tensor(unsw_scaled,    dtype=torch.float32)
charles_tensor = torch.tensor(charles_scaled, dtype=torch.float32)

unsw_loader    = DataLoader(TensorDataset(unsw_tensor),    batch_size=128,
shuffle=True)
charles_loader = DataLoader(TensorDataset(charles_tensor), batch_size=128,
shuffle=True)

# Model definitions
class Autoencoder(nn.Module):
    def __init__(self, input_dim, latent_dim=32):
        super().__init__()
        self.encoder = nn.Sequential(nn.Linear(input_dim, 64), nn.ReLU(),
nn.Linear(64, latent_dim))
        self.decoder = nn.Sequential(nn.Linear(latent_dim, 64), nn.ReLU(),
nn.Linear(64, input_dim))
    def forward(self, x):
        z = self.encoder(x)
        return self.decoder(z), z

class Discriminator(nn.Module):
    def __init__(self, latent_dim=32):
        super().__init__()
        self.model = nn.Sequential(nn.Linear(latent_dim, 64), nn.ReLU(),
nn.Linear(64, 1), nn.Sigmoid())
    def forward(self, z):
        return self.model(z)

latent_dim     = 32
unsw_ae        = Autoencoder(unsw_scaled.shape[1],    latent_dim)
charles_ae     = Autoencoder(charles_scaled.shape[1], latent_dim)
discriminator  = Discriminator(latent_dim)

criterion_recon = nn.MSELoss()
criterion_adv   = nn.BCELoss()
opt_unsw_ae      = optim.Adam(unsw_ae.parameters(),      lr=1e-3)
opt_charles_ae   = optim.Adam(charles_ae.parameters(),   lr=1e-3)
opt_discriminator = optim.Adam(discriminator.parameters(), lr=1e-3)

# Adversarial training loop
epochs = 50
for epoch in range(epochs):
```

```
    unsw_ae.train(); charles_ae.train(); discriminator.train()
    total_recon_loss = total_discriminator_loss = total_generator_loss = 0

    for (unsw_batch,), (charles_batch,) in zip(unsw_loader, charles_loader):
        bs_unsw    = unsw_batch.size(0)
        bs_charles = charles_batch.size(0)

        # Train discriminator
        opt_discriminator.zero_grad()
        _, unsw_z    = unsw_ae(unsw_batch)
        _, charles_z = charles_ae(charles_batch)
        loss_d = (criterion_adv(discriminator(unsw_z.detach()),
torch.ones(bs_unsw, 1)) +
                  criterion_adv(discriminator(charles_z.detach()),
torch.zeros(bs_charles, 1))) / 2
        loss_d.backward(); opt_discriminator.step()

        # Train autoencoders
        opt_unsw_ae.zero_grad(); opt_charles_ae.zero_grad()
        recon_unsw,    z_unsw    = unsw_ae(unsw_batch)
        recon_charles, z_charles = charles_ae(charles_batch)
        loss_recon = criterion_recon(recon_unsw, unsw_batch) +
criterion_recon(recon_charles, charles_batch)
        loss_adv   = (criterion_adv(discriminator(z_unsw),    torch.zeros(bs_unsw,
1)) +
                      criterion_adv(discriminator(z_charles),
torch.ones(bs_charles, 1))) / 2
        loss_ae = loss_recon + 0.1 * loss_adv
        loss_ae.backward(); opt_unsw_ae.step(); opt_charles_ae.step()
```

2.2 Embedding Extraction and Cosine Similarity Enrichment

```
# Extract latent embeddings
unsw_ae.eval(); charles_ae.eval()
with torch.no_grad():
    _, unsw_embeddings    = unsw_ae(unsw_tensor)
    unsw_embeddings       = unsw_embeddings.cpu().numpy()
    _, charles_embeddings = charles_ae(charles_tensor)
    charles_embeddings    = charles_embeddings.cpu().numpy()

# Dynamic enrichment via cosine similarity matching
def dynamic_enrichment_similarity(unsw_embeddings, charles_embeddings):
    enriched_embeddings = []
    for i, unsw_emb in enumerate(unsw_embeddings):
        similarities    = cosine_similarity([unsw_emb], charles_embeddings)[0]
        best_match_idx  = np.argmax(similarities)
        enriched_emb    = np.concatenate([unsw_emb,
charles_embeddings[best_match_idx]])
        enriched_embeddings.append(enriched_emb)
    return np.array(enriched_embeddings)

enriched_unsw_embeddings = dynamic_enrichment_similarity(unsw_embeddings,
charles_embeddings)

df_enriched_unsw = pd.DataFrame(enriched_unsw_embeddings)
df_enriched_unsw['label'] = unsw_numeric['label'].values
```

2.3 t-SNE Visualization

```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
```

```
n_samples = 100
unsw_sample_idx    = np.random.choice(len(unsw_embeddings),    n_samples,
replace=False)
charles_sample_idx = np.random.choice(len(charles_embeddings), n_samples,
replace=False)

embeddings_before = np.vstack([unsw_embeddings[unsw_sample_idx],
charles_embeddings[charles_sample_idx]])
labels_before     = np.array([0]*n_samples + [1]*n_samples)

tsne = TSNE(n_components=2, random_state=42)
embeddings_2d_before = tsne.fit_transform(embeddings_before)

enriched_sample    = enriched_unsw_embeddings[unsw_sample_idx]
tsne_after         = TSNE(n_components=2, random_state=42)
embeddings_2d_after = tsne_after.fit_transform(enriched_sample)

fig, axs = plt.subplots(1, 2, figsize=(12, 5))
scatter = axs[0].scatter(embeddings_2d_before[:, 0], embeddings_2d_before[:, 1],
                         c=labels_before, cmap='coolwarm', alpha=0.7)
axs[0].set_title('Original t-SNE embeddings (100 samples per dataset)')
axs[0].legend(handles=scatter.legend_elements()[0], labels=['UNSW', 'Charles'])
axs[1].scatter(embeddings_2d_after[:, 0], embeddings_2d_after[:, 1], c='green',
alpha=0.7)
axs[1].set_title('Enriched t-SNE embeddings (UNSW, 100 samples)')
plt.tight_layout(); plt.show()
```

2.4 Class Balancing

```
# Stratified undersampling to balance classes
df_majority = df_enriched_unsw[df_enriched_unsw['label'] == 1]
df_minority = df_enriched_unsw[df_enriched_unsw['label'] == 0]
n_target    = 30000

df_majority_sampled = df_majority.sample(n=n_target,
replace=len(df_majority)<n_target, random_state=42)
df_minority_sampled = df_minority.sample(n=n_target,
replace=len(df_minority)<n_target, random_state=42)

df_enriched_unsw = pd.concat([df_majority_sampled, df_minority_sampled])
df_enriched_unsw = df_enriched_unsw.sample(frac=1,
random_state=42).reset_index(drop=True)
```

3. UNSW-NB15 — Classification

3.1 DFNN with Enriched Data

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, confusion_matrix, roc_curve
from sklearn.preprocessing import StandardScaler

X = df_enriched_unsw.drop(columns=['label']).values
y = df_enriched_unsw['label'].values
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_temp,  X_test,  y_temp,  y_test  = train_test_split(X, y, test_size=0.2,
random_state=42)
X_train, X_val,   y_train, y_val   = train_test_split(X_temp, y_temp,
test_size=0.2, random_state=42)

class DFNN(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 256), nn.LeakyReLU(0.1),  nn.Dropout(0.5),
            nn.Linear(256, 128),       nn.LeakyReLU(0.01), nn.Dropout(0.5),
            nn.Linear(128, 64),        nn.LeakyReLU(0.01), nn.Dropout(0.5),
            nn.Linear(64, 1),          nn.Sigmoid()
        )
    def forward(self, x):
        return self.model(x)

def train_evaluate_metrics(X_train, y_train, X_val, y_val, X_test, y_test,
input_dim, epochs=100, patience=7):
    model     = DFNN(input_dim)
    criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-5, weight_decay=1e-5)

    train_loader = DataLoader(
        TensorDataset(torch.tensor(X_train, dtype=torch.float32),
                      torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)),
        batch_size=128, shuffle=True)

    val_tensor = torch.tensor(X_val, dtype=torch.float32)
    val_labels = torch.tensor(y_val, dtype=torch.float32).unsqueeze(1)
    train_losses, val_losses = [], []
    best_val_loss = float('inf')
    epochs_no_improve = 0

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for batch_X, batch_y in train_loader:
            optimizer.zero_grad()
            loss = criterion(model(batch_X), batch_y)
            loss.backward(); optimizer.step()
            running_loss += loss.item()
        avg_train_loss = running_loss / len(train_loader)
```

```
        train_losses.append(avg_train_loss)

        model.eval()
        with torch.no_grad():
            val_loss = criterion(model(val_tensor), val_labels).item()
            val_losses.append(val_loss)

        if val_loss < best_val_loss - 1e-4:
            best_val_loss = val_loss; epochs_no_improve = 0
        else:
            epochs_no_improve += 1
            if epochs_no_improve >= patience:
                break

    model.eval()
    with torch.no_grad():
        outputs = model(torch.tensor(X_test,
dtype=torch.float32)).numpy().flatten()
        preds   = (outputs > 0.5).astype(int)

    metrics = {
        'accuracy':  accuracy_score(y_test, preds),
        'precision': precision_score(y_test, preds),
        'recall':    recall_score(y_test, preds),
        'f1_score':  f1_score(y_test, preds),
        'roc_auc':   roc_auc_score(y_test, outputs)
    }
    return model, metrics, preds, y_test, outputs, train_losses, val_losses

model, metrics, preds, y_true, probas, train_losses, val_losses =
train_evaluate_metrics(
    X_train, y_train, X_val, y_val, X_test, y_test, input_dim=X.shape[1],
epochs=50, patience=7
)
```

3.2 Cross-Validation — Enriched UNSW-NB15

```
from sklearn.model_selection import StratifiedKFold

kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_metrics_list = []

for fold, (train_idx, test_idx) in enumerate(kf.split(X, y), 1):
    X_train_full, X_test_cv = X[train_idx], X[test_idx]
    y_train_full, y_test_cv = y[train_idx], y[test_idx]

    val_split  = int(0.8 * len(X_train_full))
    X_train_cv = X_train_full[:val_split]; X_val_cv = X_train_full[val_split:]
    y_train_cv = y_train_full[:val_split]; y_val_cv = y_train_full[val_split:]

    _, fold_metrics, _, _, _, _, _ = train_evaluate_metrics(
        X_train_cv, y_train_cv, X_val_cv, y_val_cv, X_test_cv, y_test_cv,
        input_dim=X.shape[1], epochs=50, patience=7
    )
    cv_metrics_list.append(fold_metrics)

avg_cv_metrics = {k: np.mean([m[k] for m in cv_metrics_list]) for k in
cv_metrics_list[0]}
```

3.3 DFNN without Enrichment — UNSW-NB15 Baseline

```
import torch, torch.nn as nn, torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score, roc_auc_score, confusion_matrix, roc_curve
from sklearn.preprocessing import StandardScaler
import numpy as np, pandas as pd

# Balance dataset via undersampling
df_features = pd.DataFrame(unsw_scaled, columns=[f'feat_{i}' for i in
range(unsw_scaled.shape[1])])
df = pd.concat([df_features, unsw_numeric['label']], axis=1)
min_class_count = df['label'].value_counts().min()
df_balanced = df.groupby('label', group_keys=False).apply(lambda x:
x.sample(min_class_count, random_state=42))

X_balanced       = df_balanced.drop(columns=['label']).values
y_balanced       = df_balanced['label'].values
scaler           = StandardScaler()
X_balanced_scaled = scaler.fit_transform(X_balanced)

X_train, X_val, y_train, y_val = train_test_split(
    X_balanced_scaled, y_balanced, test_size=0.2, random_state=42,
stratify=y_balanced)

train_loader = DataLoader(
    TensorDataset(torch.tensor(X_train, dtype=torch.float32),
                  torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)),
    batch_size=128, shuffle=True)
val_tensor = torch.tensor(X_val,  dtype=torch.float32)
val_labels = torch.tensor(y_val,  dtype=torch.float32).unsqueeze(1)
val_dataset   = TensorDataset(val_tensor, val_labels)
val_loader    = DataLoader(val_dataset, batch_size=128, shuffle=False)
train_dataset = TensorDataset(torch.tensor(X_train, dtype=torch.float32),
                              torch.tensor(y_train,
dtype=torch.float32).unsqueeze(1))

class DFNN(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 256), nn.LeakyReLU(0.01), nn.Dropout(0.1),
            nn.Linear(256, 128),       nn.LeakyReLU(0.01), nn.Dropout(0.1),
            nn.Linear(128, 64),        nn.LeakyReLU(0.01), nn.Dropout(0.1),
            nn.Linear(64, 1),          nn.Sigmoid()
        )
    def forward(self, x): return self.model(x)

model     = DFNN(input_dim=unsw_scaled.shape[1])
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5, weight_decay=1e-5)

# Training with early stopping
epochs = 100; patience = 5; best_val_loss = np.inf; early_stop_counter = 0
train_losses, val_losses = [], []

for epoch in range(epochs):
    model.train()
    total_train_loss = 0
    for x_batch, y_batch in train_loader:
        optimizer.zero_grad()
        loss = criterion(model(x_batch), y_batch)
        loss.backward(); optimizer.step()
```

```
        total_train_loss += loss.item() * x_batch.size(0)
    train_losses.append(total_train_loss / len(train_dataset))

    model.eval(); total_val_loss = 0
    with torch.no_grad():
        for x_batch, y_batch in val_loader:
            total_val_loss += criterion(model(x_batch), y_batch).item() *
x_batch.size(0)
    avg_val_loss = total_val_loss / len(val_dataset)
    val_losses.append(avg_val_loss)

    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss; best_model_state = model.state_dict();
early_stop_counter = 0
    else:
        early_stop_counter += 1
        if early_stop_counter >= patience: break

model.load_state_dict(best_model_state)
```

4. CIC-IDS2017 — Data Preparation and Enrichment

4.1 Data Loading and Preprocessing

```
import pandas as pd, numpy as np, glob
import torch
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

files_path = '/content/drive/MyDrive/0. B/2024/MLE/Proyectos Activos 2024/Cristian
- Profiling/Data/CICIDS2017/*.csv'
cicids = pd.concat([pd.read_csv(f) for f in glob.glob(files_path)],
ignore_index=True)
cicids.columns = cicids.columns.str.strip()

# Binary label
cicids['binary_label'] = cicids['Label'].apply(lambda x: 0 if x == 'BENIGN' else 1)

# Stratified 5% subset
cicids_subset, _ = train_test_split(cicids, test_size=0.95,
stratify=cicids['binary_label'], random_state=42)

# Numeric features — impute and scale
cicids_numeric = cicids_subset.select_dtypes(include=['int64','float64']).copy()
cicids_numeric = cicids_numeric.replace([np.inf, -np.inf], np.nan)
cicids_numeric = cicids_numeric.fillna(cicids_numeric.median())
scaler_cic     = StandardScaler()
cicids_scaled  = scaler_cic.fit_transform(cicids_numeric)

cicids_tensor  = torch.tensor(cicids_scaled, dtype=torch.float32)
labels_tensor  = torch.tensor(cicids_subset['binary_label'].values,
dtype=torch.long)
cicids_loader  = DataLoader(TensorDataset(cicids_tensor), batch_size=128,
shuffle=True)
```

4.2 Adversarial Autoencoder Training

```
import torch.nn as nn, torch.optim as optim
from sklearn.metrics.pairwise import cosine_similarity

charles_numeric = charles_data.select_dtypes(include=['int64','float64']).copy()
scaler_charles  = StandardScaler()
charles_scaled  = scaler_charles.fit_transform(charles_numeric)
charles_tensor  = torch.tensor(charles_scaled, dtype=torch.float32)
charles_loader  = DataLoader(TensorDataset(charles_tensor), batch_size=128,
shuffle=True)

# Autoencoder and Discriminator classes (same architecture as UNSW-NB15)
class Autoencoder(nn.Module):
    def __init__(self, input_dim, latent_dim=32):
        super().__init__()
        self.encoder = nn.Sequential(nn.Linear(input_dim, 64), nn.ReLU(),
nn.Linear(64, latent_dim))
        self.decoder = nn.Sequential(nn.Linear(latent_dim, 64), nn.ReLU(),
nn.Linear(64, input_dim))
    def forward(self, x):
        z = self.encoder(x); return self.decoder(z), z

class Discriminator(nn.Module):
    def __init__(self, latent_dim=32):
```

```
        super().__init__()
        self.model = nn.Sequential(nn.Linear(latent_dim, 64), nn.ReLU(),
nn.Linear(64, 1), nn.Sigmoid())
    def forward(self, z): return self.model(z)

latent_dim     = 32
cicids_ae      = Autoencoder(cicids_tensor.shape[1],  latent_dim)
charles_ae     = Autoencoder(charles_tensor.shape[1], latent_dim)
discriminator  = Discriminator(latent_dim)
criterion_recon = nn.MSELoss(); criterion_adv = nn.BCELoss()
opt_cicids_ae   = optim.Adam(cicids_ae.parameters(),      lr=1e-3)
opt_charles_ae  = optim.Adam(charles_ae.parameters(),     lr=1e-3)
opt_discriminator = optim.Adam(discriminator.parameters(), lr=1e-3)

epochs = 20
for epoch in range(epochs):
    cicids_ae.train(); charles_ae.train(); discriminator.train()
    total_recon = total_disc = total_adv = 0

    for (cic_batch,), (charles_batch,) in zip(cicids_loader, charles_loader):
        bs_cic  = cic_batch.size(0); bs_char = charles_batch.size(0)

        # Discriminator step
        opt_discriminator.zero_grad()
        _, z_cic  = cicids_ae(cic_batch); _, z_char = charles_ae(charles_batch)
        loss_d = (criterion_adv(discriminator(z_cic.detach()),  torch.ones(bs_cic,
1)) +
                  criterion_adv(discriminator(z_char.detach()),
torch.zeros(bs_char, 1))) / 2
        loss_d.backward(); opt_discriminator.step()

        # Autoencoder step
        opt_cicids_ae.zero_grad(); opt_charles_ae.zero_grad()
        recon_cic, z_cic   = cicids_ae(cic_batch)
        recon_char, z_char = charles_ae(charles_batch)
        loss_recon = criterion_recon(recon_cic, cic_batch) +
criterion_recon(recon_char, charles_batch)
        loss_adv   = (criterion_adv(discriminator(z_cic),  torch.zeros(bs_cic,  1))
+
                      criterion_adv(discriminator(z_char), torch.ones(bs_char,
1))) / 2
        loss_ae = loss_recon + 0.1 * loss_adv
        loss_ae.backward(); opt_cicids_ae.step(); opt_charles_ae.step()
```

4.3 Embedding Extraction and Enrichment

```
cicids_ae.eval(); charles_ae.eval()
with torch.no_grad():
    _, cic_emb  = cicids_ae(cicids_tensor); cic_emb  = cic_emb.cpu().numpy()
    _, char_emb = charles_ae(charles_tensor); char_emb = char_emb.cpu().numpy()

def dynamic_enrichment_similarity(cic_emb, char_emb):
    enriched = []
    for i, e in enumerate(cic_emb):
        sims    = cosine_similarity([e], char_emb)[0]
        best_idx = np.argmax(sims)
        enriched.append(np.concatenate([e, char_emb[best_idx]]))
    return np.array(enriched)

enriched_cicids = dynamic_enrichment_similarity(cic_emb, char_emb)

df_enriched_cicids = pd.DataFrame(enriched_cicids)
```

```
df_enriched_cicids['label'] = cicids_subset['binary_label'].values
```

4.4 t-SNE Visualization

```
from sklearn.manifold import TSNE

n_samples         = 100
cicids_sample_idx = np.random.choice(len(cic_emb),  n_samples, replace=False)
charles_sample_idx = np.random.choice(len(char_emb), n_samples, replace=False)

embeddings_before  = np.vstack([cic_emb[cicids_sample_idx],
char_emb[charles_sample_idx]])
labels_before      = np.array([0]*n_samples + [1]*n_samples)

embeddings_2d_before = TSNE(n_components=2, random_state=42,
perplexity=30).fit_transform(embeddings_before)
embeddings_2d_after  = TSNE(n_components=2, random_state=42,
perplexity=30).fit_transform(enriched_cicids[cicids_sample_idx])

fig, axs = plt.subplots(1, 2, figsize=(12, 5))
scatter = axs[0].scatter(embeddings_2d_before[:,0], embeddings_2d_before[:,1],
c=labels_before, cmap='coolwarm', alpha=0.7)
axs[0].set_title('Original t-SNE embeddings (100 samples per dataset)')
axs[0].legend(handles=scatter.legend_elements()[0], labels=['CICIDS','Charles'])
axs[1].scatter(embeddings_2d_after[:,0], embeddings_2d_after[:,1], c='green',
alpha=0.7)
axs[1].set_title('Enriched t-SNE embeddings (CICIDS, 100 samples)')
plt.tight_layout(); plt.show()
```

5. CIC-IDS2017 — Classification

5.1 DFNN with Enriched Data

```
import torch, torch.nn as nn, torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, confusion_matrix, roc_curve
from sklearn.preprocessing import StandardScaler

X = df_enriched_cicids.drop(columns=['label']).values
y = df_enriched_cicids['label'].values
scaler = StandardScaler(); X = scaler.fit_transform(X)

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.2,
random_state=42, stratify=y_temp)

class DFNN(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 256), nn.LeakyReLU(0.1),  nn.Dropout(0.5),
            nn.Linear(256, 128),       nn.LeakyReLU(0.01), nn.Dropout(0.5),
            nn.Linear(128, 64),        nn.LeakyReLU(0.01), nn.Dropout(0.5),
            nn.Linear(64, 1),          nn.Sigmoid()
        )
    def forward(self, x): return self.model(x)

def train_evaluate_metrics(X_train, y_train, X_val, y_val, X_test, y_test,
input_dim, epochs=50, patience=7):
    model = DFNN(input_dim); criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-5, weight_decay=1e-5)
    train_loader = DataLoader(
        TensorDataset(torch.tensor(X_train, dtype=torch.float32),
                      torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)),
        batch_size=128, shuffle=True)
    val_tensor = torch.tensor(X_val, dtype=torch.float32)
    val_labels = torch.tensor(y_val, dtype=torch.float32).unsqueeze(1)
    train_losses, val_losses = [], []; best_val_loss = float('inf'); no_improve = 0

    for epoch in range(epochs):
        model.train(); running_loss = 0.0
        for batch_X, batch_y in train_loader:
            optimizer.zero_grad(); loss = criterion(model(batch_X), batch_y)
            loss.backward(); optimizer.step(); running_loss += loss.item()
        train_losses.append(running_loss / len(train_loader))
        model.eval()
        with torch.no_grad():
            val_loss = criterion(model(val_tensor), val_labels).item()
            val_losses.append(val_loss)
        if val_loss < best_val_loss - 1e-4:
            best_val_loss = val_loss; no_improve = 0
        else:
            no_improve += 1
            if no_improve >= patience: break

    model.eval()
    with torch.no_grad():
```

```
        outputs = model(torch.tensor(X_test,
dtype=torch.float32)).numpy().flatten()
        preds   = (outputs > 0.5).astype(int)
    return model, {
        'accuracy':  accuracy_score(y_test, preds),  'precision':
precision_score(y_test, preds),
        'recall':    recall_score(y_test, preds),     'f1_score':  f1_score(y_test,
preds),
        'roc_auc':   roc_auc_score(y_test, outputs)
    }, preds, y_test, outputs, train_losses, val_losses

model, metrics, preds, y_true, probas, train_losses, val_losses =
train_evaluate_metrics(
    X_train, y_train, X_val, y_val, X_test, y_test, input_dim=X.shape[1]
)
```

5.2 DFNN without Enrichment — CIC-IDS2017 Baseline

```
# Balance dataset via undersampling
X = cicids_scaled; y = cicids_subset['binary_label'].values
df_features = pd.DataFrame(X, columns=[f'feat_{i}' for i in range(X.shape[1])])
df = pd.concat([df_features, pd.Series(y, name="label")], axis=1)
min_class_count = df['label'].value_counts().min()
df_balanced = df.groupby('label', group_keys=False).apply(lambda x:
x.sample(min_class_count, random_state=42))
X_balanced = df_balanced.drop(columns=['label']).values
y_balanced = df_balanced['label'].values

X_train, X_val, y_train, y_val = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced)

train_loader = DataLoader(
    TensorDataset(torch.tensor(X_train, dtype=torch.float32),
                  torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)),
    batch_size=128, shuffle=True)
val_tensor = torch.tensor(X_val, dtype=torch.float32)
val_labels = torch.tensor(y_val, dtype=torch.float32).unsqueeze(1)
val_loader    = DataLoader(TensorDataset(val_tensor, val_labels), batch_size=128)
train_dataset = TensorDataset(torch.tensor(X_train, dtype=torch.float32),
                              torch.tensor(y_train,
dtype=torch.float32).unsqueeze(1))
val_dataset   = TensorDataset(val_tensor, val_labels)

class DFNN(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 256), nn.LeakyReLU(0.01), nn.Dropout(0.1),
            nn.Linear(256, 128),       nn.LeakyReLU(0.01), nn.Dropout(0.1),
            nn.Linear(128, 64),        nn.LeakyReLU(0.01), nn.Dropout(0.1),
            nn.Linear(64, 1),          nn.Sigmoid()
        )
    def forward(self, x): return self.model(x)

model = DFNN(input_dim=X_balanced.shape[1])
criterion = nn.BCELoss(); optimizer = optim.Adam(model.parameters(), lr=1e-5,
weight_decay=1e-5)
epochs = 50; patience = 5; best_val_loss = np.inf; early_stop_counter = 0
train_losses, val_losses = [], []

for epoch in range(epochs):
    model.train(); total_train_loss = 0
```

```
    for x_batch, y_batch in train_loader:
        optimizer.zero_grad(); loss = criterion(model(x_batch), y_batch)
        loss.backward(); optimizer.step(); total_train_loss += loss.item() *
x_batch.size(0)
    train_losses.append(total_train_loss / len(train_dataset))

    model.eval(); total_val_loss = 0
    with torch.no_grad():
        for x_batch, y_batch in val_loader:
            total_val_loss += criterion(model(x_batch), y_batch).item() *
x_batch.size(0)
    avg_val_loss = total_val_loss / len(val_dataset);
val_losses.append(avg_val_loss)

    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss; best_model_state = model.state_dict();
early_stop_counter = 0
    else:
        early_stop_counter += 1
        if early_stop_counter >= patience: break

model.load_state_dict(best_model_state)
```

6. Gaussian Noise Baseline

6.1 Noise Enrichment

```
# Append Gaussian noise as surrogate behavioral features
n_samples = cicids_scaled.shape[0]
noise_dim  = 54
noise      = np.random.normal(0, 1, size=(n_samples, noise_dim))
enriched_cicids_noise = np.hstack([cicids_scaled, noise])

df_enriched_cicids = pd.DataFrame(enriched_cicids_noise)
df_enriched_cicids['label'] = cicids_subset['binary_label'].values
```

6.2 t-SNE Visualization — Noise Baseline

```
from sklearn.manifold import TSNE

idx_sample        = np.random.choice(len(cicids_scaled), 100, replace=False)
baseline_sample   = cicids_scaled[idx_sample]
enriched_sample   = enriched_cicids_noise[idx_sample]

embeddings_2d_before = TSNE(n_components=2, random_state=42,
perplexity=30).fit_transform(baseline_sample)
embeddings_2d_after  = TSNE(n_components=2, random_state=42,
perplexity=30).fit_transform(enriched_sample)

fig, axs = plt.subplots(1, 2, figsize=(12, 5))
axs[0].scatter(embeddings_2d_before[:,0], embeddings_2d_before[:,1], c='blue',
alpha=0.7)
axs[0].set_title('Baseline t-SNE embeddings (100 samples)')
axs[1].scatter(embeddings_2d_after[:,0],  embeddings_2d_after[:,1],  c='green',
alpha=0.7)
axs[1].set_title('t-SNE embeddings with Gaussian noise (100 samples)')
plt.tight_layout(); plt.show()
```

6.3 DFNN Classification with Gaussian Noise Enrichment

```
X = df_enriched_cicids.drop(columns=['label']).values
y = df_enriched_cicids['label'].values
scaler = StandardScaler(); X = scaler.fit_transform(X)

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.2,
random_state=42, stratify=y_temp)

# DFNN and training function identical to Section 5.1
def train_evaluate(X_train, y_train, X_val, y_val, X_test, y_test, input_dim,
epochs=50, patience=7):
    model = DFNN(input_dim); criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-5, weight_decay=1e-5)
    train_loader = DataLoader(
        TensorDataset(torch.tensor(X_train, dtype=torch.float32),
                      torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)),
        batch_size=128, shuffle=True)
    val_tensor = torch.tensor(X_val, dtype=torch.float32)
    val_labels = torch.tensor(y_val, dtype=torch.float32).unsqueeze(1)
    train_losses, val_losses = [], []; best_val_loss = float('inf'); no_improve =
0; best_state = None
```

```
    for epoch in range(epochs):
        model.train(); running_loss = 0
        for xb, yb in train_loader:
            optimizer.zero_grad(); loss = criterion(model(xb), yb)
            loss.backward(); optimizer.step(); running_loss += loss.item()
        train_losses.append(running_loss / len(train_loader))
        model.eval()
        with torch.no_grad():
            val_loss = criterion(model(val_tensor), val_labels).item()
            val_losses.append(val_loss)
        if val_loss < best_val_loss - 1e-4:
            best_val_loss = val_loss; no_improve = 0; best_state =
model.state_dict()
        else:
            no_improve += 1
            if no_improve >= patience: break

    model.load_state_dict(best_state); model.eval()
    with torch.no_grad():
        probs = model(torch.tensor(X_test, dtype=torch.float32)).numpy().flatten()
        preds = (probs > 0.5).astype(int)
    return model, {
        'accuracy':  accuracy_score(y_test, preds),  'precision':
precision_score(y_test, preds),
        'recall':    recall_score(y_test, preds),     'f1':        f1_score(y_test,
preds),
        'roc_auc':   roc_auc_score(y_test, probs)
    }, preds, y_test, probs, train_losses, val_losses

model, metrics, preds, y_true, probs, train_losses, val_losses = train_evaluate(
    X_train, y_train, X_val, y_val, X_test, y_test, input_dim=X.shape[1]
)
```

7. Permutation Feature Importance

```
from sklearn.metrics import roc_auc_score
import numpy as np

def permutation_importance(model, X_test, y_test, metric_fn, n_repeats=5):
    baseline_score = metric_fn(y_test,
        model(torch.tensor(X_test,
dtype=torch.float32)).detach().numpy().flatten())
    importances = np.zeros(X_test.shape[1])
    for col in range(X_test.shape[1]):
        scores = []
        for _ in range(n_repeats):
            X_permuted = X_test.copy()
            np.random.shuffle(X_permuted[:, col])
            score = metric_fn(y_test,
                model(torch.tensor(X_permuted,
dtype=torch.float32)).detach().numpy().flatten())
            scores.append(baseline_score - score)
        importances[col] = np.mean(scores)
    return importances

feature_importances = permutation_importance(model, X_test, y_test, roc_auc_score,
n_repeats=3)
feature_importances = feature_importances / np.sum(feature_importances)

n_features_total    = X.shape[1]
n_noise             = 54
n_traffic           = n_features_total - n_noise

traffic_importances = feature_importances[:n_traffic]
noise_importances   = feature_importances[n_traffic:]
```

Bibliography

[Abiramasundari-Ramaswamy-25] S. Abiramasundari and V. Ramaswamy, “DDoS attack detection using supervised machine learning algorithms,” *Scientific Reports*, vol. 15, p. 13098, 2025. https://doi.org/10.1038/s41598-024-84879-y

[Aggarwal-16] C. C. Aggarwal, *Recommender Systems: The Textbook*. Springer, 2016. https://link.springer.com/book/9783319296579

[Almuhammadi-AlBakhat-22] S. Almuhammadi and L. Al-Bakhat, “Intrusion detection on QUIC traffic: A machine learning approach,” in *Proc. 7th Int. Conf. Data Science and Machine Learning Applications (CDMA)*, Riyadh, Saudi Arabia, 2022. https://doi.org/10.1109/CDMA54072.2022.00037

[Alshehri-Khan-23] A. Alshehri et al., “Cyberattack detection framework using machine learning and user behavior analytics,” *Computer Systems Science and Engineering*, vol. 44, pp. 1679–1689, 2023. https://doi.org/10.32604/csse.2023.026526

[Altulaihan-Almaiah-24] E. Altulaihan, M. A. Almaiah, and A. Aljughaiman, “Anomaly detection IDS for detecting DoS attacks in IoT networks based on machine learning algorithms,” *Sensors*, vol. 24, p. 713, 2024. https://doi.org/10.3390/s24020713

[Aouedi-Le-25] O. Aouedi et al., “Deep learning on network traffic prediction: Recent advances, analysis, and future directions,” *ACM Computing Surveys*, vol. 57, pp. 1–37, 2025. https://doi.org/10.1145/3703447

[Bashurov-Safonov-23] V. Bashurov and P. Safonov, “Anomaly detection in network traffic using entropy-based methods,” *Issues of Information Systems*, vol. 24, pp. 82–94, 2023. https://doi.org/10.48009/4_iis_2023_107

[Bengio-13] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013. https://arxiv.org/pdf/1206.5538

[Benova-Hudec-24] L. Benova and L. Hudec, “Comprehensive analysis and evaluation of anomalous user activity in web server logs,” *Sensors*, vol. 24, p. 746, 2024. https://doi.org/10.3390/s24030746

[Cerf-Kahn-74] V. G. Cerf and R. E. Kahn, “A protocol for packet network intercommunication,” *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637–648, 1974. https://apps.dtic.mil/sti/tr/pdf/ADA634240.pdf

[Cover-Thomas-06] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley, 2006.

[Davies-Bouldin-79] D. L. Davies and D. W. Bouldin, “A cluster separation measure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–228, 1979. https://doi.org/10.1109/TPAMI.1979.4766909

[Denning-87] D. E. Denning, “An intrusion-detection model,” *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, Feb. 1987. https://doi.org/10.1109/TSE.1987.232894

[Duan-Fu-23] X. Duan, Y. Fu, and K. Wang, “Network traffic anomaly detection method based on multi-scale residual classifier,” *Computer Communications*, vol. 198, pp. 206–216, 2023. https://doi.org/10.1016/j.comcom.2022.10.024

[Elsayed-Mohamed-24] S. Elsayed, K. Mohamed, and M. A. Madkour, “A comparative study of using deep learning algorithms in network intrusion detection,” *IEEE Access*, vol. 12, pp. 58851–58870, 2024. https://doi.org/10.1109/ACCESS.2024.3389096

[Ester-96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining (KDD)*, Portland, OR, 1996, pp. 226–231.

[Garroppo-Giardina-25] R. Garroppo et al., “Trustworthy AI and federated learning for intrusion detection in 6G-connected smart buildings,” *Future Internet*, vol. 17, p. 191, 2025. https://doi.org/10.3390/fi17050191

[Goodfellow-Bengio-Courville-16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. Available: https://www.deeplearningbook.org

[Greenacre-Groenen-22] M. Greenacre et al., “Principal component analysis,” *Nature Reviews Methods Primers*, vol. 2, p. 100, 2022. https://doi.org/10.1038/s43586-022-00184-w

[He-Yin-21] J. He and X. Yin, “Internet user behavior analysis based on big data,” in *Proc. IWCMC*, Harbin, China, 2021, pp. 432–435. https://doi.org/10.1109/IWCMC51323.2021.9498875

[Hooshmand-Hosahalli-22] M. K. Hooshmand and D. Hosahalli, “Network anomaly detection using deep learning techniques,” *CAAI Transactions on Intelligence Technology*, vol. 7, pp. 228–243, 2022. https://doi.org/10.1049/cit2.12078

[Hooshmand-Huchaiah-24] M. K. Hooshmand et al., “Robust network anomaly detection using ensemble learning approach and explainable artificial intelligence (XAI),” *Alexandria Engineering Journal*, vol. 94, pp. 120–130, 2024. https://doi.org/10.1016/j.aej.2024.03.041

[Ikotun-Ezugwu-23] A. M. Ikotun et al., “K-means clustering algorithms: A comprehensive review,” *Information Sciences*, vol. 622, pp. 178–210, 2023. https://doi.org/10.1016/j.ins.2022.11.139

[ISO-84] International Organization for Standardization, *Information Processing Systems — Open Systems Interconnection — Basic Reference Model*, ISO 7498, 1984.

[Javed-Awais-24] A. Javed et al., “Embedding tree-based intrusion detection system in smart thermostats for enhanced IoT security,” *Sensors*, vol. 24, p. 7320, 2024. https://doi.org/10.3390/s24227320

[Kalnoor-Gowrishankar-21] G. Kalnoor and S. Gowrishankar, “IoT-based smart environment using intelligent intrusion detection system,” *Soft Computing*, vol. 25, pp. 11573–11588, 2021. https://doi.org/10.1007/s00500-021-06028-1

[Kamal-Mashaly-25] H. Kamal and M. Mashaly, “Enhanced hybrid deep learning models-based anomaly detection method for two-stage binary and multi-class classification of attacks in intrusion detection systems,” *Algorithms*, vol. 18, p. 69, 2025. https://doi.org/10.3390/a18020069

[Kasongo-Sun-20] S. M. Kasongo and Y. Sun, “Performance analysis of intrusion detection systems using UNSW-NB15,” *Journal of Big Data*, vol. 7, p. 105, 2020. https://doi.org/10.1186/s40537-020-00379-6

[Lakhina-04] A. Lakhina, M. Crovella, and C. Diot, “Diagnosing network-wide traffic anomalies,” in *Proc. ACM SIGCOMM*, Portland, OR, 2004, pp. 219–230. https://doi.org/10.1145/1030194.1015492

[Li-Rios-21] Z. Li, A. L. G. Rios, and L. Trajković, “Machine learning for detecting anomalies and intrusions in communication networks,” *IEEE Journal on Selected Areas in Communications*, vol. 39, pp. 2254–2264, 2021. https://ieeexplore.ieee.org/abstract/document/9430511

[MacQueen-67] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, Berkeley, CA, 1967, pp. 281–297.

[Marfo-Tosh-24] W. Marfo, D. K. Tosh, and S. V. Moore, “Enhancing network anomaly detection using graph neural networks,” in *Proc. 22nd Mediterranean Communication and Computer Networking Conference (MedComNet)*, Nice, France, 2024, pp. 1–10. https://ieeexplore.ieee.org/abstract/document/10578278

[Martin-Fernandez-21] A. G. Martín et al., “A survey for user behavior analysis based on machine learning techniques: Current models and applications,” *Applied Intelligence*, vol. 51, pp. 6029–6055, 2021. https://doi.org/10.1007/s10489-020-02160-x

[Marx-24] V. Marx, “Seeing data as t-SNE and UMAP do,” *Nature Methods*, vol. 21, pp. 930–933, 2024. https://doi.org/10.1038/s41592-024-02301-x

[Mitchell-99] T. M. Mitchell, “Machine learning and data mining,” *Communications of the ACM*, vol. 42, no. 11, pp. 30–36, 1999.

[Modell-Larson-21] A. Modell et al., “A graph embedding approach to user behavior anomaly detection,” in *Proc. IEEE Int. Conf. Big Data*, Orlando, FL, USA, 2021, pp. 2650–2655. https://doi.org/10.1109/BigData52589.2021.9671423.

[Moustafa-Slay-15] N. Moustafa and J. Slay, “UNSW-NB15: A comprehensive data set for network intrusion detection systems,” in *Proc. MilCIS*, Canberra, Australia, 2015, pp. 1–6. https://doi.org/10.1109/MilCIS.2015.7348942

[Muntean-Militaru-23] M. Muntean and F. D. Militaru, “Metrics for evaluating classification algorithms,” in *Proc. IE 2022*, 2023, pp. 307–317. https://doi.org/10.1007/978-981-19-6755-9_24

[Murthy-Asghar-24] A. Murthy, M. R. Asghar, and W. Tu, “A lightweight intrusion detection for Internet of Things-based smart buildings,” *Security and Privacy*, vol. 7, p. e386, 2024. https://doi.org/10.1002/spy2.386

[Oran-Kocak-22] S. Oran and A. Koçak, “Security review and performance analysis of QUIC and TCP protocols,” in *Proc. 15th Int. Conf. Information Security and Cryptography (ISCTURKEY)*, Ankara, Turkey, 2022. https://doi.org/10.1109/ISCTURKEY56345.2022.9931821

[P-Palaniappan-24] S. P. et al., “Dynamic behavioral profiling for anomaly detection in software-defined IoT networks,” *Peer-to-Peer Networking and Applications*, vol. 17, pp. 2450–2469, 2024. https://doi.org/10.1007/s12083-024-01694-y

[Pandhurnekar-Iyyappan-23] V. Pandhurnekar et al., “Proposed method for threat detection using user behavior analysis,” in *Proc. IEEE TEMSMET*, Mysuru, India, 2023, pp. 1–5. https://doi.org/10.1109/TEMSMET56707.2023.10150053

[Rani-Gill-23] D. Rani et al., “Design of an intrusion detection model for IoT-enabled smart home,” *IEEE Access*, vol. 11, pp. 52509–52526, 2023. https://doi.org/10.1109/ACCESS.2023.3276863

[Rosay-Cheval-22] A. Rosay et al., “Network intrusion detection: A comprehensive analysis of CIC-IDS2017,” in *Proc. ICISSP*, 2022, pp. 25–36. https://univ-lemans.hal.science/hal-03563228/

[Salem-Stolfo-11] M. B. Salem and S. J. Stolfo, “Masquerade attack detection using a search-behavior modeling approach,” Columbia University, New York, NY, Tech. Rep. CUCS-027-09, 2011. https://academiccommons.columbia.edu/doi/10.7916/D8X63TSV

[Sen-Dasgupta-23] J. Sen and S. Dasgupta, “Data privacy preservation on the Internet of Things,” *arXiv preprint arXiv:2304.00258*, 2023. https://doi.org/10.5772/intechopen.111477

[Shafi-Lashkari-24] M. Shafi et al., “NTLFlowLyzer: Intrusion detection dataset and behavior profiling,” *Computers & Security*, vol. 148, p. 104160, 2024. https://doi.org/10.1016/j.cose.2024.104160

[Shahapure-Nicholas-20] K. R. Shahapure and C. Nicholas, “Cluster quality analysis using silhouette score,” in *Proc. IEEE DSAA*, 2020, pp. 747–748. https://doi.org/10.1109/DSAA49011.2020.00096.

[Shannon-48] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[Shirdel-DiMauro-24] M. Shirdel et al., “Worthiness benchmark for classification metrics,” *Information Sciences*, vol. 678, p. 120882, 2024. https://doi.org/10.1016/j.ins.2024.120882

[Singh-Girdhar-22] H. V. Singh, A. Girdhar, and S. Dahiya, “A literature survey based on DBSCAN algorithms,” in *Proc. 6th Int. Conf. Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2022, pp. 751–758. https://doi.org/10.1109/ICICCS53718.2022.9788440.

[Singh-Jindal-23] I. Singh and R. Jindal, “Outlier based intrusion detection in databases for user behaviour analysis,” *International Journal of Machine Learning and Cybernetics*, vol. 15, pp. 2573–2593, 2023. https://doi.org/10.1007/s13042-023-02049-4

[Tao-Lu-24] X. Tao et al., “User behavior threat detection based on adaptive sliding window GAN,” *IEEE Transactions on Network and Service Management*, vol. 21, pp. 2493–2503, 2024. https://doi.org/10.1109/TNSM.2024.3355698

[Tokuda-Comin-22] E. K. Tokuda et al., “Revisiting agglomerative clustering,” *Physica A*, vol. 585, p. 126433, 2022. https://doi.org/10.1016/j.physa.2021.126433

[Trevisan-Giordano-20] M. Trevisan et al., “Five years at the edge: Watching internet from the ISP network,” *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 561–574, 2020. https://doi.org/10.1109/TNET.2020.2967588

[Tun-Nyaung-20] M. T. Tun, D. E. Nyaung, and M. P. Phyu, “Network anomaly detection using threshold-based sparse,” in *Proc. 11th Int. Conf. Advances in Information Technology*, Bangkok, Thailand, 2020, pp. 1–8. https://doi.org/10.1145/3406601.3406626

[Vapnik-95] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY: Springer, 1995.

[Wang-Balarezo-21] S. Wang et al., “Machine learning in network anomaly detection: A survey,” *IEEE Access*, vol. 9, pp. 152379–152396, 2021. https://doi.org/10.1109/ACCESS.2021.3126834

[Wang-Jones-20] L. Wang and R. Jones, “Big data analytics in cyber security: Network traffic and attacks,” *Journal of Computer Information Systems*, vol. 61, pp. 410–417, 2020. https://doi.org/10.1080/08874417.2019.1688731

[Wang-Yang-23] M. Wang, N. Yang, and N. Weng, “Securing a smart home with a transformer-based IoT intrusion detection system,” *Electronics*, vol. 12, p. 2100, 2023. https://doi.org/10.3390/electronics12092100

[Xi-Zhou-22] Z. Xi et al., “Newton: Intent-driven network traffic monitoring,” *IEEE/ACM Transactions on Networking*, vol. 30, pp. 939–952, 2022. https://doi.org/10.1109/TNET.2021.3128557

[Yu-22] Y. Yu, “Big data network security defense mode of deep learning algorithm,” *Open Computer Science*, vol. 12, pp. 345–356, 2022. https://doi.org/10.1515/comp-2022-0257

[Zhao-Guo-24] Z. Zhao et al., “A multi-information fusion anomaly detection model,” *Scientific Reports*, vol. 14, p. 16147, 2024. https://doi.org/10.1038/s41598-024-66760-0

Author Index

Abiramasundari 92

Aggarwal 17

Almuhammadi 2

Alshehri 89, 91

Altulaihan 1

Aouedi 1

Bashurov 45, 89

Bengio 15

Benova 88

Cerf 9, 10, 43

Cover 18

Davies 34

Denning 2, 12

Duan 90

Elsayed 1

Ester 17, 34

Garroppo 2

Goodfellow 15, 32, 44, 61, 66, 67

Greenacre 13, 30

He 88, 91

Hooshmand 1, 19, 42–45

Ikotun 88

ISO 9

Javed 1

Kalnoor 1
Kamal 1
Kasongo 92
Lakhina 11
Li 19
MacQueen 17, 34
Marfo 92
Martín 2, 19, 21, 44
Marx 57
Mitchell 14
Modell 89, 92
Moustafa 2, 47
Muntean 68
Murthy 1
Oran 2
P., S. 89, 92
Pandhurnekar 89, 92
Rani 2, 19
Rosay 2, 48
Salem 19–21, 29, 42, 45
Sen 40
Shafi 89, 92
Shahapure 34
Shannon 30, 45
Shirdel 68

Singh 21, 42, 88, 89
Tao 2, 19, 21
Tokuda 34
Trevisan 1, 2, 11, 19
Tun 90, 92
Vapnik 14, 16
Wang 1, 2, 19, 45, 89
Xi 90
Yu 1
Zhao 1

Subject Index

A

ablation study…55–60, 72, 76, 79, 83, 85, 90–92, 94
accuracy 1, 4, 6, 16, 55–57, 68, 70–76, 78–83, 86, 87, 90, 91, 94
activation function.....65, 66
adversarial alignment. 2, 6, 50, 51, 55, 57, 60, 70, 74, 81, 87, 90–92
adversarial training.....43, 51, 52, 63, 93
agglomerative clustering.....34–38
anomaly detection 1–3, 5, 6, 8, 13, 14, 16, 17, 20, 21, 36, 38–40, 42–48, 50, 54, 58, 60–90, 92–95
attack categories.....47–49
autoencoder. .6, 15, 21, 31–33, 35–39, 43, 50–53, 55, 57, 60, 63, 70, 72, 74, 79, 81, 86, 88, 92, 93

B

backdoor.....45
balanced accuracy.....4, 68, 72–74, 79–81, 86
baseline model. …3, 55–57, 62, 63, 66, 68, 69, 71–74, 76, 78–83, 86, 87, 90, 92, 94
behavioral clustering.....28, 33
behavioral dataset.3, 5, 21–23, 25, 28, 30, 31, 34, 38–42, 50, 51, 58, 63–65, 91, 92
behavioral embedding....2–4, 6, 8, 28, 33, 36, 38–40, 43, 46, 50, 53–58, 61–65, 69, 79, 83, 84, 87–93
behavioral feature.....8, 20, 28, 30–33, 37, 39
behavioral profile…2, 3, 6, 12, 17–22, 28, 34–39, 45, 50, 53–59, 61, 64, 88–90
behavioral profiling.....5, 7, 19–41, 58, 92–94
behavioral representation…21, 22, 25, 28, 30, 31, 33, 42, 43, 50, 52, 53
behavioral vector.....3, 20, 21, 34, 89
benchmark dataset.....6, 46, 48, 63, 86, 92
binary classification.....47, 51, 52
binary cross-entropy.....52, 65, 67, 86
bootstrap.....4, 68, 84
Brier score.....68, 74, 81, 86

C

Charles Proxy.3, 7, 22–24, 36, 40, 41, 51, 60, 65
CIC-IDS2017. 2, 4, 42, 46, 48, 49, 52, 53, 55–58, 63, 69, 76–83, 85–87, 90, 91, 93–95
classification metrics.....6, 68, 73, 80, 87
cluster quality.....34, 38, 93
clustering..6, 8, 13–15, 17, 20–22, 24, 28, 33–39, 88, 93
confidence cascade.57, 60, 83–85, 87, 90–92, 94, 95
confusion matrix.....70–72, 78, 79, 86, 87
cookie.....23, 40
cosine similarity. 3, 13, 17, 43, 50, 52–55, 60, 63, 93
cross-domain enrichment. 2, 8, 36, 40, 42–60, 63, 87, 91
cross-validation.....69, 74–76, 81, 82
cybersecurity.....19, 48

D

data collection.....6–8, 22–25, 27, 40
data imbalance.....1, 16
data leakage2, 3, 7, 55, 69, 71–75, 77–80, 82, 83, 90, 92–94
Davies–Bouldin index.....34, 35, 38
DBSCAN.....17, 34, 37
deep feedforward neural network (DFNN)..8, 32, 61, 62, 65, 66, 70, 71, 76–78, 82, 83, 86, 91, 93
deep learning..1, 3, 5–8, 21, 43, 60–87, 92, 93
denial of service (DoS).....47, 49
dimensionality reduction....13, 14, 21, 30, 38, 39, 92
discriminator.....51, 52, 55, 56, 93
dropout.....65, 67

E

embedding....2–4, 6, 8, 28, 33, 35, 36, 38–40, 43, 46, 50–58, 60–65, 69, 79, 83–93
encoder.....15, 31–33, 50–53, 64
enriched dataset....6, 8, 40, 42, 43, 50, 54, 60, 61, 69, 70, 77

enriched feature vector...8, 43, 57, 58, 60–64, 66, 67, 94
entropy. 16, 29–31, 39, 45, 52, 65, 67, 86, 88, 93
epoch.....4, 33, 52, 65–67, 69, 70, 77, 78, 94
ethics and consent.....24, 40, 41
explainability.....1, 53, 92, 95
exploit attack.....45, 47

F

F1-score.....6, 16, 68, 74–76, 81, 82, 86
false negative.....16, 68, 71, 78, 79
false positive.....16, 68, 71, 78, 79
feature extraction.....44, 46, 48, 62, 63
feature selection.....1
feature vector6, 13, 16, 17, 21, 31–33, 37, 43, 44, 46–48, 50–52, 54, 61–64, 66, 67, 86, 89, 92, 93
federated learning.....1

G

generalization.....70, 76, 85, 91, 95
generic attack.....47
gradient descent.....16, 67
graph neural network.....1, 92

H

HTTP/HTTPS.....7, 10, 11, 20, 22–24, 28, 40
hybrid model.....36–39, 46
hypothesis.....3, 4, 87

I

interaction pattern 3, 5, 10, 22, 25, 29, 30, 33–36, 39, 50, 54, 89, 90
intrusion detection system (IDS). 1, 8, 44, 46, 48, 92
IoT (Internet of Things).....1

K

K-means clustering.....34, 37, 41

L

latent representation…3, 6, 14, 15, 20, 21, 31–33, 35, 37–39, 43, 50–54, 57, 63–65, 89, 93
latent space5–7, 15, 31, 32, 37–39, 43, 51, 53, 58, 63, 64, 89, 90, 93
learning curve.....69, 70, 77

M

machine learning....1, 7, 8, 13, 14, 17, 21, 28, 44, 61, 87, 92
Mann–Whitney U test.....53
Matthews correlation coefficient (MCC)4, 68, 74, 81, 86
memorization.....90
monitoring..1, 2, 10, 19, 20, 22–25, 28, 43, 88

N

network analyst.....6, 24, 25, 28, 93
network anomaly detection..2, 3, 5, 8, 43, 45, 46, 50, 60–87
network flow 5, 10, 11, 47, 48, 50, 57, 61, 63, 86, 91
network security.....7, 14, 19–42, 45
network traffic..2–6, 8–12, 17, 20, 22, 36, 38–40, 42–65, 69, 76, 85, 87–89, 93, 94
neural network…8, 31, 32, 44, 61, 62, 65–67, 76, 82, 83, 86, 87
noise control.....4, 55, 56, 84, 85, 91
normalization.....17, 28–30, 50, 93

O

overfitting.....16, 67

P

packet. 5, 9–13, 19, 22, 23, 25, 39, 40, 43–45, 47, 48, 50, 54, 59, 62, 63, 65, 88–90, 92
paired t-test.....4, 68, 74, 81, 84
PCA (principal component analysis)…30, 31, 39, 55, 56, 88, 93
PR-AUC.....68, 74, 81, 86
precision.....4, 6, 16, 68, 72–74, 76, 79–82, 86
proxy traffic log 28, 35, 36, 38, 39, 41, 51, 58, 64, 69

R

recall.6, 16, 68, 72–74, 76, 79–82, 85, 86, 91, 95
reconnaissance attack.....45, 47
reconstruction error.....15, 32, 51
reconstruction loss.....32, 52
ReLU.....33, 65, 66
representation learning.5, 7, 8, 15, 21, 22, 28, 30–33, 38–40, 42, 60, 88, 92, 94, 95

ROC AUC 4, 55–57, 68, 71–76, 78–87, 91, 94

S

semantic validity 53, 94
session 6, 9–11, 20–24, 28, 31, 36, 39–41, 47, 48, 62, 88, 91, 93
shellcode attack 45, 47
sigmoid function 65, 66
silhouette score 34–39, 93
specificity 68, 72–74, 80, 81, 86
synthetic control 55, 57, 58, 83, 94

T

t-SNE visualization 55, 57, 58
traffic analysis . . 10, 12, 13, 17, 19, 22, 45, 88, 94
traffic features . . 2–6, 8, 19, 38, 40, 43, 47–49, 58, 59, 61–63, 71, 79, 83, 86, 88, 89, 93
training loss 69, 77
transformer 1, 71, 72, 76, 86
true negative 68, 71
true positive 16, 68, 71

U

UNSW-NB15 . 2, 4, 42, 46–48, 52, 53, 55, 56, 58, 63, 69–76, 78, 80, 81, 83–87, 90–95
URL 23, 40, 41
user behavior . . 2–8, 11, 12, 19–42, 45, 50, 59, 60, 71, 77, 88, 91, 93
user behavioral profiling 2, 3, 5, 7, 19–41, 45, 93
user groups 6, 24, 25, 37–39, 93
user session 21, 23, 28, 31, 36, 39, 88

V

validation loss 65, 67, 69, 70, 77

W

web browsing 10, 11, 23, 88
web service 24, 28
worm propagation 45, 47
worm propagation 26, 29