

الجمهورية الجزائرية الديمقراطية الشعبية
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزارة التعليم العالي و البحث العلمي
Ministry of Higher Education and Scientific Research
– تلمسان – جامعة أبي بكر بلقايد
Aboubakr Belkaïd University – Tlemcen –
Faculty of Technology



MASTER'S THESIS

Submitted in partial fulfillment of the requirements for the **Master's degree**

In: Telecommunications

Specialty: Networks and Telecommunications

By Omar REZMA and Mohammed Ziyad Miloud KERNABI

Topic

**Deep Learning-Based Anomaly
Detection in Enterprise Networks**

Publicly defended on 24/06/2026, in front of the examination committee:

S. KAMECHE	Full Professor	University of Tlemcen	Chair
N. BOUKLI-HACENE	Full Professor	University of Tlemcen	Examiner
F. DERRAZ	Associate Professor A	University of Tlemcen	Supervisor
M. BOUSAHLA	Associate Professor B	University of Tlemcen	Co-supervisor

Academic Year : 2025–2026

Dedication

I dedicate this work to my beloved family, whose unwavering love, encouragement, and sacrifices have been the foundation of my success. Their constant support and belief in my abilities have motivated me throughout this academic journey. I also dedicate this work to my dear friends, who have stood by my side, offering encouragement, understanding, and valuable support during both the challenging and rewarding moments of this project. To all of you, I express my deepest gratitude and appreciation.

REZMA Omar

Dedication

I dedicate this work to: My beloved parents, for their unconditional love, sacrifices, unwavering support, and encouragement throughout my academic journey. My dear grandparents , whose wisdom, prayers, kindness, and affection have always been a source of inspiration and strength. My beloved sisters, for their love, encouragement, understanding, and continuous support throughout this journey. My family, for their presence, trust, and moral support. My teachers, who shared their knowledge and contributed significantly to my education and personal development. My friends and colleagues, for their help, motivation, and the memorable moments we shared throughout this journey. Finally, to all those who contributed, directly or indirectly, to the completion of this work.

Mohammed Ziyad Miloud KERNABI

Acknowledgements

First and foremost, we thank God Almighty for giving us the strength, patience, and determination required to complete this work.

We would like to express our sincere gratitude to our supervisor, Mr. F. DERRAZ, and our co-supervisor, Mr. M. BOUSAHLA, for their guidance, availability, and valuable advice throughout the preparation of this thesis. Their support helped us improve the scientific quality and practical relevance of this work.

We also extend our thanks to the members of the examination committee for taking the time to evaluate this thesis and for their constructive remarks and suggestions.

Our appreciation goes to the faculty members of the Department of Telecommunications for the knowledge, skills, and academic support they provided during our studies.

Finally, we would like to thank our families and friends for their continuous encouragement, patience, and support throughout this academic journey. To everyone who contributed directly or indirectly to the completion of this work, we express our sincere thanks.

Abstract

The increasing frequency and sophistication of cyberattacks have made network security a major concern for modern information systems. Intrusion Detection Systems (IDS) play a crucial role in identifying malicious activities and protecting network infrastructures from potential threats. However, traditional IDS solutions often struggle to detect complex attack patterns and to adapt to the growing volume and diversity of network traffic. To address these challenges, this thesis proposes a hybrid deep learning model that combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks for network intrusion detection.

The proposed model was developed and evaluated using the NSL-KDD dataset. A comprehensive preprocessing pipeline was applied, including data cleaning, categorical feature encoding, feature scaling, and feature selection. The CNN component was used to automatically extract relevant traffic features, while the LSTM component captured sequential dependencies and temporal patterns. The original feature space was reduced from 41 features to 20 informative features in order to improve computational efficiency and facilitate embedded deployment.

Experimental evaluation on the KDDTest+ dataset produced an accuracy of 80.5%, a precision of 96.8%, a recall of 68.0%, and an F1-score of 79.9%. These results demonstrate the model's ability to generate highly reliable intrusion alerts while maintaining a reasonable attack detection capability. In addition, the trained model was successfully deployed on a Raspberry Pi within a controlled network environment. The deployment experiments confirmed the feasibility of performing real-time intrusion detection on low-cost embedded hardware.

The findings highlight the effectiveness of hybrid CNN–LSTM architectures for network intrusion detection and demonstrate their potential for practical cybersecurity applications requiring both accurate detection and real-time operation.

Keywords: Intrusion Detection System, IDS, Deep Learning, CNN–LSTM, NSL-KDD, Cybersecurity, Network Security, Raspberry Pi, Real-Time Detection.

Résumé

La fréquence et la sophistication croissantes des cyberattaques font de la sécurité des réseaux une préoccupation majeure pour les systèmes d'information modernes. Les systèmes de détection d'intrusion (IDS) jouent un rôle essentiel dans l'identification des activités malveillantes et la protection des infrastructures réseau. Cependant, les solutions IDS traditionnelles peinent souvent à détecter des schémas d'attaque complexes et à s'adapter à la diversité croissante du trafic réseau. Pour répondre à ces difficultés, ce mémoire propose un modèle hybride d'apprentissage profond combinant des réseaux de neurones convolutifs (CNN) et des réseaux à mémoire court et long terme (LSTM) pour la détection d'intrusions dans les réseaux.

Le modèle proposé a été développé et évalué à l'aide du jeu de données NSL-KDD. Une chaîne complète de prétraitement a été appliquée, comprenant le nettoyage des données, l'encodage des variables catégorielles, la normalisation des caractéristiques et la sélection des variables les plus pertinentes. La composante CNN a permis d'extraire automatiquement les caractéristiques utiles du trafic, tandis que la composante LSTM a capturé les dépendances séquentielles et les motifs temporels. L'espace de caractéristiques initial a été réduit de 41 à 20 variables informatives afin d'améliorer l'efficacité de calcul et de faciliter un déploiement embarqué.

L'évaluation expérimentale sur l'ensemble KDDTest+ a permis d'obtenir une exactitude (accuracy) de 80,5 %, une précision de 96,8 %, un rappel de 68,0 % et un score F1 de 79,9 %. Ces résultats démontrent la capacité du modèle à générer des alertes d'intrusion fiables tout en conservant une capacité de détection raisonnable. De plus, le modèle entraîné a été déployé avec succès sur un Raspberry Pi au sein d'un environnement réseau contrôlé. Les expériences de déploiement ont confirmé la faisabilité d'une détection d'intrusion en temps réel sur un matériel embarqué à faible coût.

Les résultats obtenus mettent en évidence l'efficacité des architectures hybrides CNN-LSTM pour la détection d'intrusions réseau et démontrent leur potentiel pour des applications pratiques de cybersécurité nécessitant à la fois une détection précise et un fonctionnement en temps réel.

Mots-clés : Système de détection d'intrusion, IDS, apprentissage profond, CNN-LSTM, NSL-KDD, cybersécurité, sécurité des réseaux, Raspberry Pi, détection en temps réel.

ملخص

أدى التزايد المستمر في عدد وتعقيد الهجمات الإلكترونية إلى جعل أمن الشبكات من أهم التحديات التي تواجه الأنظمة المعلوماتية الحديثة. وتُعد أنظمة كشف التسلل من الوسائل الأساسية المستخدمة لحماية الشبكات من خلال اكتشاف الأنشطة غير المشروعة ومحاولات الاختراق. إلا أن الأساليب التقليدية في كشف التسلل غالبًا ما تواجه صعوبات في التعرف على الهجمات المعقدة وتحقيق مستويات مرتفعة من الدقة، مما يستدعي الاستفادة من تقنيات الذكاء الاصطناعي والتعلم العميق لتحسين أداء هذه الأنظمة.

تهدف هذه الدراسة إلى تطوير نموذج هجين يعتمد على الشبكات العصبية الالتفافية وشبكات الذاكرة طويلة وقصيرة المدى من أجل تعزيز كفاءة كشف التسلل في الشبكات الحاسوبية. وقد تم تدريب النموذج وتقييمه باستخدام مجموعة بيانات NSL-KDD مع تطبيق سلسلة من عمليات المعالجة المسبقة للبيانات شملت تنظيف البيانات، وترميز الخصائص الفئوية، وتقييم البيانات، واختيار الخصائص الأكثر أهمية. وقد استُخدمت الشبكات العصبية الالتفافية لاستخراج الخصائص المهمة من بيانات حركة المرور الشبكية، بينما استُخدمت شبكات LSTM لاكتشاف العلاقات الزمنية والأنماط المتتابة داخل هذه البيانات.

تم تقييم أداء النموذج باستخدام مجموعة من المقاييس القياسية، حيث حقق دقة عامة بلغت 80.5%، ودقة إيجابية بلغت 96.8%، ومعدل استرجاع قدره 68.0%، ومقياس F1 بلغ 79.9%. وتُظهر هذه النتائج قدرة النموذج على تقليل الإنذارات الكاذبة بشكل كبير مع المحافظة على مستوى جيد من اكتشاف الهجمات.

بالإضافة إلى ذلك، تم نشر النموذج المُدرَّب على منصة Pi Raspberry ضمن بيئة شبكية حقيقية، وأظهرت نتائج النشر العملي قدرة النظام على مراقبة حركة الشبكة واكتشاف الأنشطة المشبوهة وإصدار التنبيهات بشكل فوري، مما يؤكد إمكانية تطبيق النموذج في البيئات الواقعية باستخدام أجهزة منخفضة التكلفة.

الكلمات المفتاحية: نظام كشف التسلل، أمن الشبكات، الأمن السيبراني، التعلم العميق، الشبكات العصبية الالتفافية، شبكات الذاكرة طويلة وقصيرة المدى، الكشف اللّاني عن الهجمات.

List of Abbreviations

CNN	Convolutional Neural Network
FN	False Negative
FP	False Positive
F1-score	Harmonic mean of Precision and Recall
GPU	Graphics Processing Unit
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
KDD	Knowledge Discovery in Databases
LAN	Local Area Network
LSTM	Long Short-Term Memory
MAN	Metropolitan Area Network
MPLS	Multiprotocol Label Switching
NSL-KDD	NSL Knowledge Discovery Dataset
RFE	Recursive Feature Elimination
RNN	Recurrent Neural Network
SD-WAN	Software-Defined Wide Area Network
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
WAN	Wide Area Network

Contents

Dedication: REZMA Omar	i
Dedication: Mohammed Ziyad Miloud KERNABI	ii
Acknowledgements	iii
Abstract	iv
ملخص	v
General Introduction	1
1 Enterprise Networks and Anomalies	4
1.1 Introduction	4
1.2 The Enterprise Network	5
1.2.1 Availability and Resilience	5
1.2.2 Scalability	5
1.2.3 Security	5
1.3 Classification and Types of Networks	6
1.3.1 Local Area Networks and VLANs	6
1.3.2 Metropolitan Area Networks	7
1.3.3 Wide Area Networks and SD-WAN	7
1.3.4 Virtual Private Networks	8
1.4 Enterprise Network Architectures	8
1.4.1 Hierarchical Architecture	9
1.4.2 Spine–Leaf Architecture	9
1.5 Network Anomalies	10
1.6 Intrusion Detection Systems	11

1.7	Relevance to Deep Learning-Based IDS	12
1.8	Conclusion	12
2	Deep Learning for Anomaly Detection	14
2.1	Introduction: From Network Security Problems to Learning Models	14
2.2	Nature of Anomalies in Enterprise Network Traffic	15
2.3	Detection Families: From Rules to Deep Models	17
2.4	Transforming Network Traffic into a Learning Problem	18
2.5	Datasets and the Meaning of Benchmark Evaluation	19
2.6	Supervised Learning Under IDS Constraints	20
2.7	Artificial Neural Networks as Trainable Decision Functions	21
2.8	Deep Representation Learning for Traffic Features	22
2.9	Convolutional Neural Networks for Feature Interaction Extraction	23
2.10	Recurrent Neural Networks and Temporal Dependence	24
2.11	Long Short-Term Memory Units	25
2.12	Hybrid CNN–LSTM Architecture for the Proposed IDS	26
2.13	Activation Functions and Their Operational Meaning	27
2.14	Loss Function, Optimizer, and Regularization	28
2.15	Decision Threshold and Alert Generation	29
2.16	Model Generalization and Evaluation Pitfalls	30
2.17	Embedded Deployment Constraints	31
2.18	Design Choices Retained our work	32
2.19	Conclusion	33
3	Datasets and Methodology	35
3.1	Introduction	35
3.2	NSL–KDD Dataset	35
3.3	Preprocessing Pipeline	36
3.3.1	Data Cleaning and Label Conversion	37
3.3.2	Categorical Encoding	38
3.3.3	Feature Scaling and selection	38
3.4	Proposed CNN–LSTM Hybrid Model	38
3.4.1	CNN Component	39
3.4.2	LSTM Component	39

3.4.3	Dense Output Layer	39
3.5	Model Training Configuration	40
3.6	Evaluation Metrics	41
3.6.1	Confusion Matrix	41
3.6.2	Accuracy	42
3.6.3	Precision	42
3.6.4	Recall	42
3.6.5	F1-score	42
3.7	Feature Selection as a Security and Deployment Step	42
3.8	Interpretability, Human Supervision, and Trust	44
3.9	Methodological Limits	45
3.10	Conclusion	45
4	Implementation and Results	47
4.1	Introduction	47
4.2	Experimental Environment	47
4.2.1	Hardware Components	47
4.2.2	Network Topology	48
4.2.3	Software Environment	49
4.2.4	Developed Software Modules	51
4.2.5	Integrated Software Pipeline	52
4.2.6	Roles of the Main Software Modules	52
4.3	Classification Results	53
4.3.1	Overall Performance Metrics	53
4.3.2	Confusion Matrix Analysis	54
4.3.3	Training Performance Analysis	54
4.3.4	Comparison of Training, Validation, and Test Metrics	55
4.4	Comparison with Related Work	56
4.5	Real-Time Detection on Raspberry Pi	56
4.5.1	Deployment Architecture	56
4.5.2	Online Experimental Workflow	58
4.5.3	Detection of Normal Traffic	59
4.5.4	Detection of Abnormal Traffic	60
4.6	Critical Analysis	61

4.6.1	Practical Interpretation of the Results	62
4.7	Conclusion	62
	General Conclusion and Perspectives	63
	Bibliography	65

List of Figures

1.1	Local Area Network and VLAN architecture.	6
1.2	Metropolitan Area Network architecture.	7
1.3	WAN and SD-WAN architecture.	8
1.4	Virtual Private Network architecture.	8
1.5	Traditional three-tier enterprise architecture.	9
1.6	Two-tier spine-leaf architecture.	10
1.7	General architecture of a network intrusion detection system.	12
2.1	Problem-driven motivation for using deep learning in enterprise anomaly de- tection.	15
2.2	Taxonomy of intrusion detection approaches and position of deep learning models.	17
2.3	Transformation of observed network events into numerical representations used by the learning model.	19
2.4	Training loop of a supervised neural intrusion detection model.	22
2.5	Flow-window matrix used to explain how selected traffic features can be or- ganized for sequence-aware learning.	23
2.6	One-dimensional convolution used to learn local interactions between se- lected traffic features.	24
2.7	LSTM gates as a controlled memory mechanism for traffic sequences. . . .	25
2.8	Problem-oriented architecture of the CNN-LSTM IDS used in this thesis. .	26
2.9	Operational IDS metrics obtained at the selected decision threshold and dis- cussed in Chapter 4.	29
2.10	Trade-off between detection quality, latency, and resource consumption in embedded IDS deployment.	32
3.1	Preprocessing pipeline used before model training and deployment.	37
3.2	Proposed CNN-LSTM hybrid IDS architecture.	40

3.3	Feature selection interpreted as both a learning step and an embedded deployment step.	43
3.4	Monitoring loop showing how deep learning alerts should remain connected to administrator analysis.	45
4.1	Hardware components used to implement the experimental platform.	48
4.2	Network topology used for the deployment and evaluation of the proposed intrusion detection system.	49
4.3	End-to-end software pipeline of the proposed IDS, from dataset preparation to online deployment.	52
4.4	Roles of the main software modules used throughout the project.	53
4.5	Confusion matrix of the CNN–LSTM model on the NSL-KDD test set.	54
4.6	Training and validation accuracy and loss curves of the proposed CNN–LSTM model.	55
4.7	Comparison of training, validation, and test performance metrics of the proposed CNN–LSTM model.	55
4.8	Initialization of the IDS application and loading of the trained CNN–LSTM model on the Raspberry Pi.	57
4.9	Real-time detection flow executed on the Raspberry Pi.	58
4.10	Experimental workflow used for live traffic generation, monitoring, and evaluation.	59
4.11	Normal traffic correctly identified by the deployed IDS.	60
4.12	Real-time detection of abnormal network traffic generated during the laboratory experiment.	61

List of Tables

2.1	Practical difficulties that make anomaly detection more complex than a simple binary classification task.	16
2.2	Comparison of detection strategies from the perspective of enterprise anomaly detection.	18
2.3	Feature representation levels used in network intrusion detection.	19
2.4	IDS datasets cited in the thesis and their methodological role.	20
2.5	Functional role of each part of the proposed CNN–LSTM architecture. . . .	27
2.6	Evaluation metrics and their operational interpretation for an IDS.	30
2.7	Synthesis of the Chapter 2 design choices and their relation to the thesis problem.	33
3.1	Description of the NSL–KDD dataset used in this thesis.	36
3.2	Summary of model hyperparameters and training configuration.	41
3.3	Feature-selection choices and their impact on the proposed IDS.	44
4.1	Software tools and frameworks used in the proposed IDS implementation. .	50
4.2	Main developed software files and their roles in the proposed IDS workflow.	51
4.3	Summary of classification performance metrics on the NSL-KDD test set. .	53
4.4	Comparison of the proposed CNN–LSTM model with related intrusion detection approaches.	56

General Introduction

Enterprise networks are now at the center of institutional, industrial, and academic information systems. They connect users, servers, applications, cloud services, embedded devices, remote workers, and security appliances through a set of heterogeneous wired and wireless technologies. Their role is no longer limited to simple data transport: they support authentication, collaborative work, database access, real-time supervision, industrial control, voice and video communication, and access to public or private cloud platforms. As a consequence, the reliability and security of these infrastructures directly affect the continuity of the services offered by an organization.

This increasing dependence on network services is accompanied by a significant growth in complexity. Modern enterprise networks often combine local area networks, VLAN segmentation, VPN access, wide-area interconnection, virtualized services, and cloud resources. This diversity generates traffic that is dynamic, high-dimensional, and sometimes difficult to characterize using simple predefined rules. In such a context, abnormal traffic may originate from several sources: cyberattacks, compromised devices, configuration errors, routing problems, hardware failures, or unexpected user behavior. Detecting these deviations at an early stage is therefore essential for protecting confidentiality, integrity, availability, and service quality.

Traditional security mechanisms, such as firewalls, access-control policies, and signature-based Intrusion Detection Systems (IDS), remain useful components of a layered defense strategy. However, they also present important limitations. Signature-based systems depend on known attack patterns and may fail when facing new or modified attacks. Purely statistical methods can detect certain deviations, but they often require strong assumptions about traffic distribution and may not capture complex nonlinear relationships between network features. The NIST guide on intrusion detection and prevention systems emphasizes that IDS and IPS technologies are designed to monitor events, identify incidents, and support response actions, but their effectiveness depends strongly on configuration, monitoring context, and the detection technique used [1].

These limitations motivate the use of learning-based approaches. Machine learning and deep learning methods are able to learn decision boundaries from data rather than relying only on manually written rules. Deep learning is particularly attractive because it can learn hierar-

chical feature representations through multiple processing layers [2, 3]. In network intrusion detection, this ability is useful because malicious behavior may not be visible through a single feature; it can appear as a combination of packet-level, flow-level, protocol-level, and temporal characteristics.

The present thesis focuses on the design, evaluation, and deployment of a deep learning-based anomaly detection system for enterprise networks. The proposed solution uses a hybrid Convolutional Neural Network and Long Short-Term Memory architecture, denoted CNN–LSTM. The CNN part acts as a feature-extraction stage that learns local relationships between the selected traffic features, while the LSTM part models sequential dependencies and preserves information over time. LSTM networks were originally introduced to address the difficulty of learning long-term dependencies in recurrent neural networks [4]. Their use in this thesis is motivated by the fact that some network attacks are not isolated events, but evolve across several successive observations.

The experimental study is based on the NSL–KDD benchmark dataset, which was proposed as an improved version of KDD Cup 99. The dataset is still widely used in intrusion detection research because it provides labeled traffic records and attack categories, while reducing several redundancy problems present in the original KDD data [5, 6]. The methodology includes data cleaning, categorical encoding, feature scaling, feature selection, model training, and performance evaluation. The selected indicators include accuracy, precision, recall, F1-score, and confusion matrix analysis, because a single metric is insufficient to evaluate an IDS. In particular, precision reflects the reliability of alerts, whereas recall reflects the ability of the system to detect attacks.

A second objective of this work is to go beyond a purely offline classification experiment. For this reason, the trained model is exported and deployed on a Raspberry Pi platform in a controlled local network environment. This embedded deployment is important because it introduces practical constraints that are not visible in a notebook-based evaluation: packet capture, feature reconstruction, preprocessing consistency, model inference, alert generation, execution time, and resource limitations. The objective is not to claim equivalence with an industrial IDS appliance, but to validate the feasibility of a low-cost embedded demonstrator for real-time traffic monitoring.

The main contributions of this thesis are summarized as follows:

- a structured presentation of enterprise network architectures, anomaly sources, and IDS constraints;
- a reformulated and referenced overview of the machine learning and deep learning concepts required to understand CNN, LSTM, and hybrid CNN–LSTM models;
- a complete preprocessing methodology for the NSL–KDD dataset, including encoding, scaling, and feature reduction from 41 original features to 20 selected features;

- the implementation and evaluation of a hybrid CNN–LSTM model for binary intrusion detection;
- the deployment of the trained model on a Raspberry Pi platform for real-time traffic classification in a laboratory network;
- a critical discussion of the obtained results, especially the trade-off between high precision and moderate recall.

This thesis is organized into four chapters. Chapter 1 introduces enterprise networks, their main architectures, and the nature of network anomalies. Chapter 2 presents the foundations of machine learning and deep learning, with emphasis on neural networks, CNNs, RNNs, LSTMs, activation functions, loss functions, and evaluation metrics. Chapter 3 describes the dataset, preprocessing pipeline, feature-selection strategy, and proposed CNN–LSTM methodology. Chapter 4 presents the implementation environment, offline evaluation, Raspberry Pi deployment, experimental observations, comparison with related work, and critical analysis. The thesis ends with a general conclusion and perspectives for future work.

Chapter 1

Enterprise Networks and Anomalies

1.1 Introduction

Enterprise networks constitute the communication infrastructure through which an organization connects its users, servers, services, applications, and external partners. In a modern organization, the network is not only a technical support layer; it is an operational resource that enables information exchange, authentication, supervision, remote access, collaboration, and access to cloud-based services. For this reason, network availability and network security have become essential requirements for business continuity.

At the same time, enterprise networks are becoming more heterogeneous. A single infrastructure may include wired Ethernet segments, wireless access, VLANs, VPN tunnels, data-center links, cloud connections, IoT devices, and remote users. Such diversity produces traffic patterns that vary according to user activity, applications, routing policies, and security controls. This variability makes anomaly detection difficult, because not all abnormal behavior corresponds to an attack and not all attacks are visible through obvious traffic peaks.

A network anomaly can be defined as a deviation from the expected behavior of the infrastructure or its traffic. It may be caused by malicious activity, such as scanning, denial of service, unauthorized access, or malware communication. It may also result from non-malicious causes, such as equipment failure, routing instability, configuration errors, overloaded services, or unusual but legitimate user activity. Consequently, an effective detection system must distinguish between normal variability and suspicious deviation.

Intrusion Detection Systems (IDS) are designed to monitor events in networks or hosts and identify signs of incidents. According to NIST SP 800-94, IDS and IPS technologies support security monitoring by analyzing events and producing alerts that can help administrators detect and respond to incidents [1]. However, traditional IDS approaches based mainly on fixed signatures are limited when attacks are unknown, modified, or hidden inside apparently legitimate traffic. These limitations explain the growing interest in anomaly-based IDS and

learning-based detection methods [7, 8, 9].

This chapter presents the network context of the thesis. It first introduces the main characteristics of enterprise networks, then describes common network types and architectures. It then discusses network anomalies and IDS families, before explaining why deep learning is relevant for modern intrusion detection.

1.2 The Enterprise Network

An enterprise network is the digital backbone of an organization. It ensures communication between users, servers, printers, industrial devices, databases, cloud services, and security appliances. Unlike a small personal network, an enterprise infrastructure is designed according to operational requirements such as availability, scalability, manageability, segmentation, and security [10, 11].

1.2.1 Availability and Resilience

Availability refers to the ability of the network to provide service when users and applications need it. In enterprise environments, service interruption can affect production, administration, communication, and customer-facing applications. To reduce the impact of failures, networks often include redundant links, backup devices, dynamic routing, link aggregation, and failover mechanisms. The goal is not to eliminate all failures, which is unrealistic, but to prevent a single failure from causing a complete service outage [12].

1.2.2 Scalability

Scalability is the ability of the infrastructure to support growth in users, devices, applications, and traffic volume without major redesign. A scalable network can be expanded by adding switches, access points, links, or virtual resources while maintaining acceptable performance. Cloud computing, virtualization, SD-WAN, and spine-leaf data-center architectures contribute to scalability because they allow more flexible allocation of resources and more predictable traffic forwarding [13, 14].

1.2.3 Security

Security aims to preserve confidentiality, integrity, and availability of information and services. Enterprise networks generally apply a defense-in-depth strategy based on authentication, access control, encryption, segmentation, monitoring, and incident response. VLANs separate broadcast domains, VPNs protect remote communications, and modern zero-trust

approaches require continuous verification of identity and device posture [15, 16]. In practice, security is not a single device but a set of coordinated mechanisms that must be continuously monitored.

1.3 Classification and Types of Networks

Enterprise networks can be classified according to their geographical coverage, communication technology, and administrative purpose. The following categories are frequently encountered in organizational infrastructures.

1.3.1 Local Area Networks and VLANs

A Local Area Network (LAN) interconnects devices within a limited area such as a building, laboratory, department, or campus. Ethernet technologies standardized by IEEE 802.3 are widely used because they provide high data rates, low latency, and mature switching mechanisms [17, 11]. LANs are suitable for internal services such as file sharing, printing, database access, and local application hosting.

Virtual Local Area Networks (VLANs) provide logical segmentation over a shared physical infrastructure. IEEE 802.1Q defines VLAN tagging, which allows switches to distinguish traffic belonging to different logical networks [18]. VLANs reduce broadcast domains, support policy enforcement, and improve security by separating users, servers, voice devices, guests, or administrative services. Figure 1.1 illustrates a typical LAN/VLAN organization.

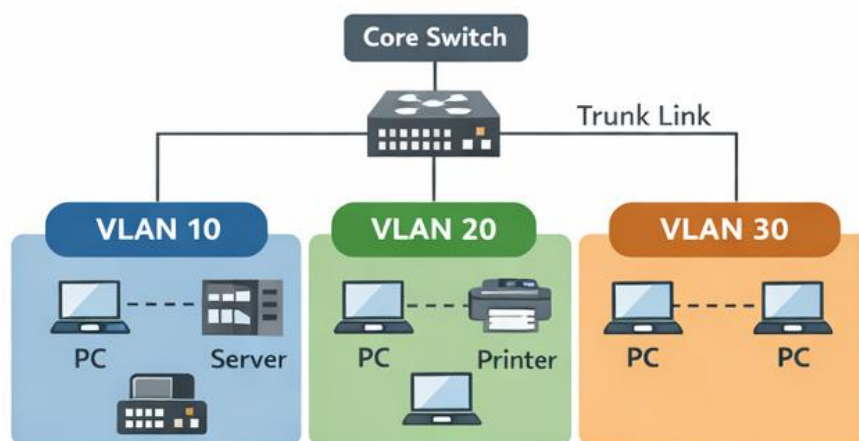


Figure 1.1: Local Area Network and VLAN architecture.

1.3.2 Metropolitan Area Networks

A Metropolitan Area Network (MAN) interconnects several local networks within the same city or metropolitan area. Such networks are used by universities, administrations, operators, and large organizations that have several sites in a limited geographical region. MAN infrastructures often rely on optical fiber links and high-capacity aggregation equipment [11]. Figure 1.2 gives a simplified view of this type of interconnection.

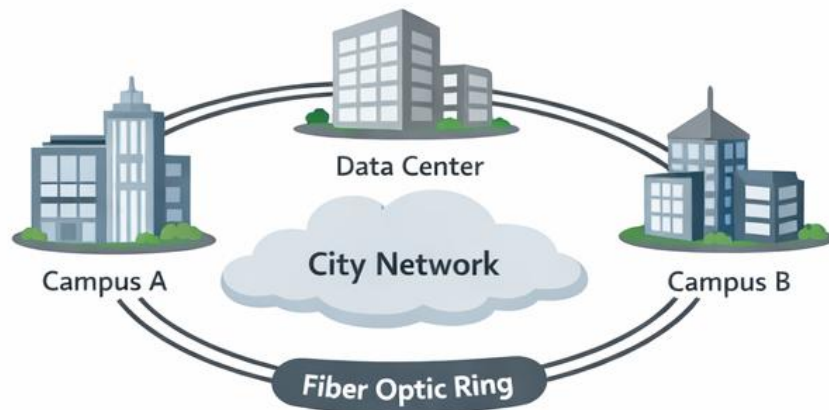


Figure 1.2: Metropolitan Area Network architecture.

1.3.3 Wide Area Networks and SD-WAN

Wide Area Networks (WANs) connect geographically distributed networks across large distances. They allow branches, campuses, data centers, and cloud services to communicate. Traditional WANs may use leased lines, MPLS, or operator-managed services [11]. Software-Defined WAN (SD-WAN) introduces centralized control and dynamic path selection over several transport links. By separating control decisions from forwarding behavior, SD-WAN can optimize traffic according to application requirements, link quality, and security policies [13]. Figure 1.3 summarizes the role of WAN and SD-WAN interconnection.

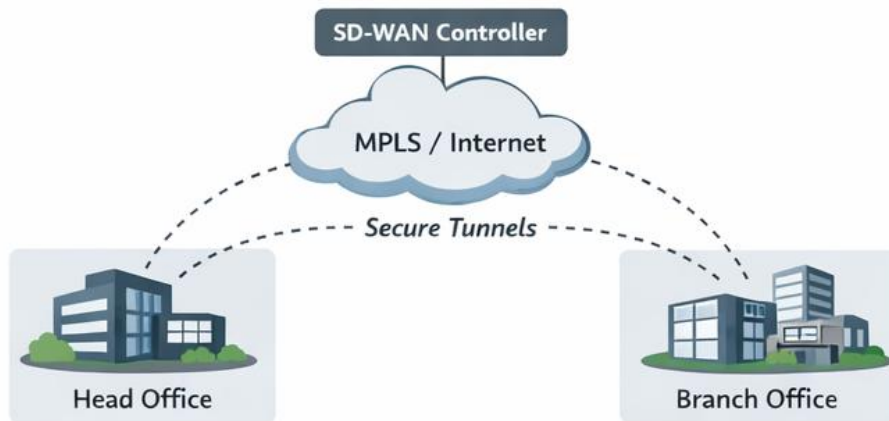


Figure 1.3: WAN and SD-WAN architecture.

1.3.4 Virtual Private Networks

A Virtual Private Network (VPN) creates a protected tunnel over an untrusted network such as the Internet. VPN mechanisms use cryptographic protocols to provide confidentiality, integrity, and authentication. In enterprise networks, VPNs are used for remote workers, inter-site communication, and secure access to internal services [15]. Figure 1.4 shows the general principle of encrypted remote access.

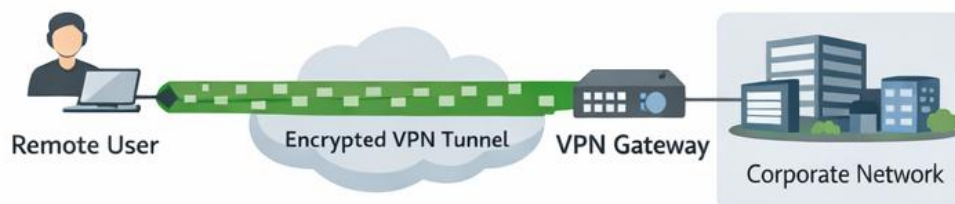


Figure 1.4: Virtual Private Network architecture.

1.4 Enterprise Network Architectures

Network architecture defines how devices, links, and services are organized. The chosen architecture influences performance, fault isolation, monitoring visibility, and the type of traffic observed by an IDS.

1.4.1 Hierarchical Architecture

The traditional hierarchical enterprise model separates the infrastructure into three layers:

- **Access layer:** connects end-user devices, phones, access points, printers, and local equipment.
- **Distribution layer:** aggregates access switches and applies routing, filtering, segmentation, and policy enforcement.
- **Core layer:** provides high-speed backbone connectivity between major network blocks.

This separation simplifies troubleshooting and makes the network easier to expand. It also allows administrators to place monitoring points at strategic positions, for example between access and distribution layers or near the core [12]. Figure 1.5 presents the three-tier architecture.

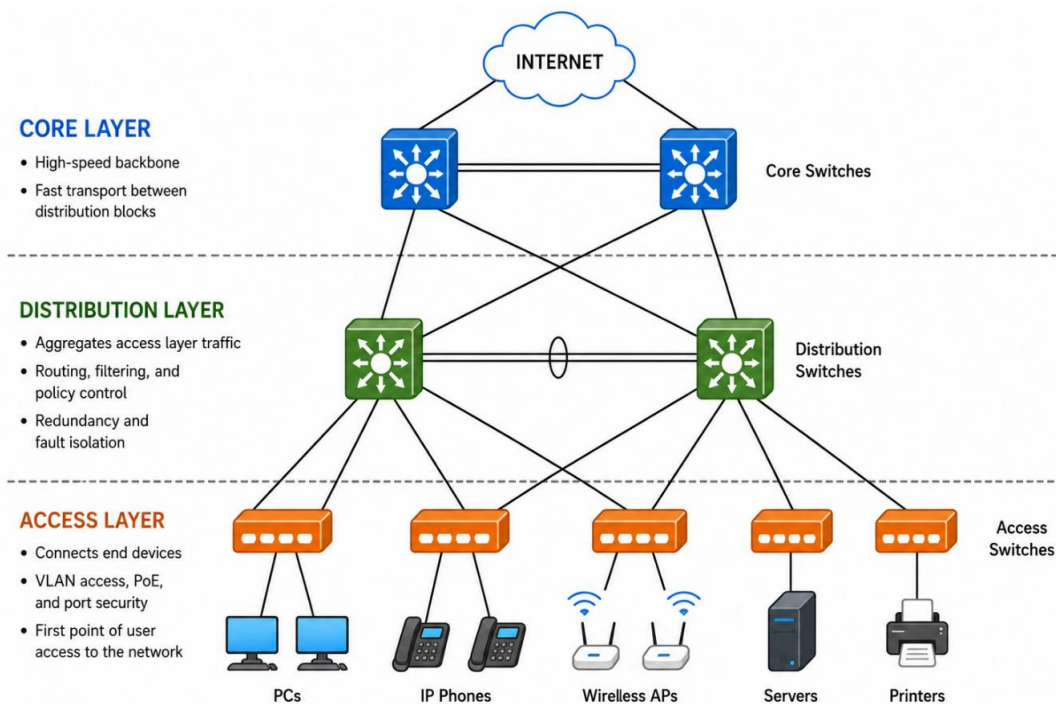


Figure 1.5: Traditional three-tier enterprise architecture.

1.4.2 Spine–Leaf Architecture

Spine–leaf architectures are widely used in data centers because they provide predictable latency and high east-west traffic capacity. Each leaf switch connects to every spine switch, creating multiple equal-cost paths. This topology is well adapted to virtualized and cloud

environments, where servers often communicate with other servers more than with external clients [14]. Figure 1.6 illustrates this architecture.

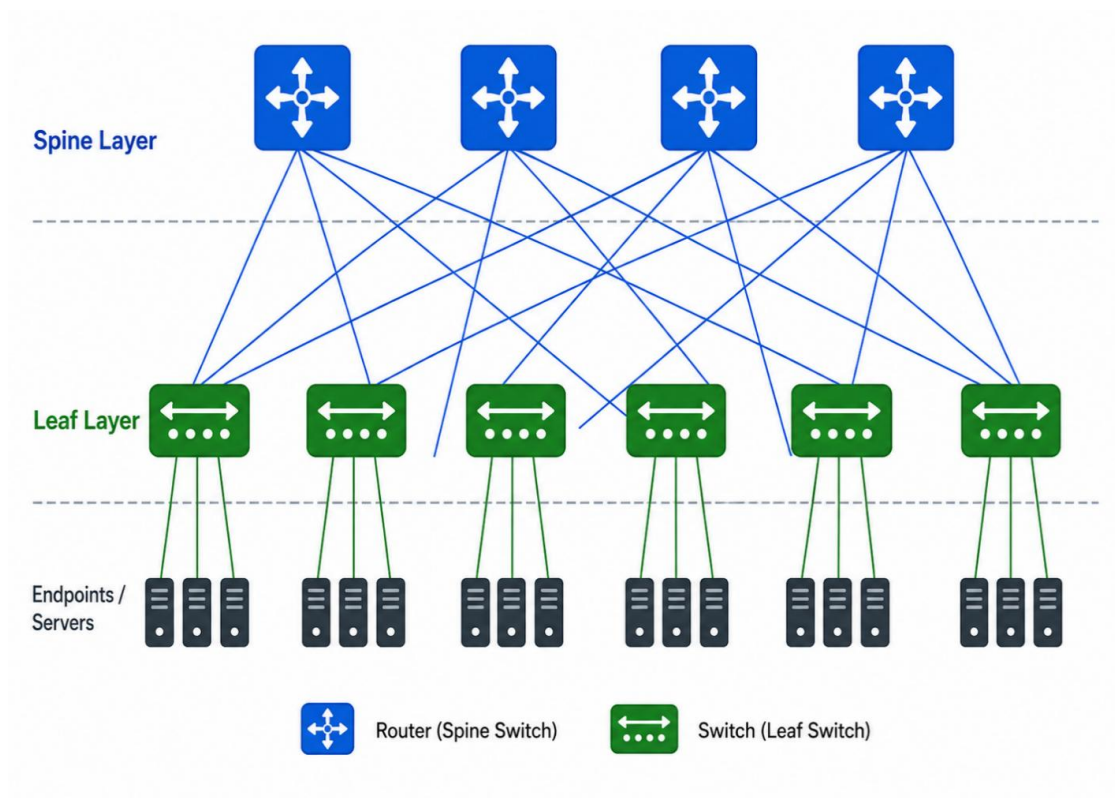


Figure 1.6: Two-tier spine-leaf architecture.

1.5 Network Anomalies

An anomaly is a behavior that differs from the expected operation of the network. In a security context, anomalies are important because they may indicate an attack, but they are also difficult to interpret because not every anomaly is malicious. For example, a sudden increase in traffic may correspond to a denial-of-service attack, but it may also be caused by a legitimate backup task or a software update campaign.

Common anomaly sources include:

- **Volumetric anomalies:** abnormal growth of traffic volume, often associated with flooding or denial-of-service behavior.
- **Protocol anomalies:** unusual packet flags, malformed fields, abnormal connection states, or unexpected protocol usage.
- **Access anomalies:** repeated login attempts, unauthorized services, unusual remote connections, or lateral movement.

- **Performance anomalies:** congestion, latency increase, packet loss, routing instability, or degraded application response.
- **Configuration anomalies:** errors introduced by routing changes, VLAN misconfiguration, firewall rules, or address conflicts.

Anomaly detection is challenging because normal traffic is not static. User behavior changes during working hours, application workloads vary, and new services can modify traffic distribution. For this reason, anomaly-based IDS must be evaluated carefully to avoid excessive false alarms [7, 19].

1.6 Intrusion Detection Systems

An Intrusion Detection System observes host or network activity in order to identify suspicious behavior. The NIST classification distinguishes several monitoring approaches, including network-based IDS, wireless IDS, network behavior analysis, and host-based IDS [1]. In this thesis, the focus is placed on network-based detection because the proposed system analyzes traffic features extracted from network flows.

IDS approaches can be grouped into three main families:

- **Signature-based detection:** compares traffic or events with known attack patterns. It is effective for known threats but weak against new or modified attacks.
- **Anomaly-based detection:** builds a model of normal behavior and reports deviations. It can detect unknown attacks but may generate false positives if normal behavior is not modeled accurately.
- **Hybrid detection:** combines signatures, anomaly analysis, and sometimes reputation or correlation mechanisms to improve detection coverage.

Figure 1.7 presents the general architecture of an IDS monitoring a network segment.

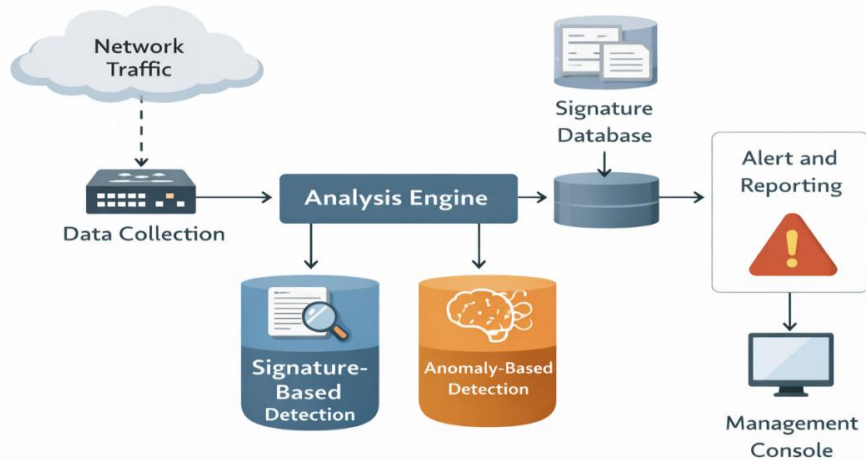


Figure 1.7: General architecture of a network intrusion detection system.

1.7 Relevance to Deep Learning-Based IDS

The diversity of enterprise network architectures generates heterogeneous traffic patterns that are difficult to describe using fixed rules. A model must learn from data, identify discriminative features, and generalize to unseen traffic. Deep learning is relevant in this context because multilayer models can learn representations at different levels of abstraction [2, 3]. In intrusion detection, CNNs can extract local feature combinations, while recurrent architectures such as LSTMs can model sequential dependencies [4, 20].

However, deep learning does not automatically solve all IDS problems. Its performance depends on dataset quality, preprocessing consistency, class distribution, evaluation protocol, and deployment constraints. An IDS with high validation accuracy may still perform poorly on new traffic if the training data is not representative. This observation motivates the methodology adopted in the following chapters: the model is trained on a benchmark dataset, evaluated using several metrics, and then deployed on a Raspberry Pi to examine practical feasibility.

1.8 Conclusion

This chapter introduced the network and security context of the thesis. Enterprise networks were presented as complex infrastructures composed of several technologies, including LANs, VLANs, WANs, VPNs, hierarchical campus models, and spine-leaf data-center architectures. The chapter also explained that anomalies may result from both malicious and non-malicious causes, which makes detection difficult in practice.

Traditional IDS techniques remain useful, but they are limited when traffic is dynamic and

attacks are unknown. This limitation supports the use of machine learning and deep learning methods for anomaly-based intrusion detection. The next chapter presents the concepts required to understand the CNN–LSTM model used in this work.

Chapter 2

Deep Learning for Anomaly Detection

2.1 Introduction: From Network Security Problems to Learning Models

The objective of this chapter is to build the theoretical foundation of the proposed intrusion detection system while keeping the discussion closely connected to the practical problem addressed in this thesis. In an enterprise network, the security problem is not limited to deciding whether a packet is authorized or forbidden. Real traffic is produced by many users, applications, servers, services, and protocols. Some activities are legitimate but unusual, while some attacks try to look as normal as possible. For this reason, anomaly detection is a difficult classification problem: the model must identify suspicious behavior without confusing every rare event with an attack. This distinction is especially important in a practical system, because an IDS that produces too many false alarms quickly becomes unusable for an administrator.

The previous chapter described the structure of enterprise networks and the role of intrusion detection systems. This chapter moves from the network view to the learning view. The main question becomes: how can traffic observations be transformed into numerical representations, and how can a model learn the difference between benign behavior and attack behavior? The answer is not only mathematical. It also involves operational choices such as the type of data used, the preprocessing strategy, the selected features, the model architecture, the decision threshold, and the deployment constraints. In this thesis, these choices are guided by the final objective: designing a CNN–LSTM based IDS that can be trained on the NSL–KDD benchmark and then deployed on a Raspberry Pi as a low-cost monitoring device.

Deep learning is attractive for intrusion detection because it can learn internal representations from data rather than relying only on hand-written rules. However, this advantage should not be interpreted as a guarantee of automatic intelligence. A deep model remains dependent on

the dataset, the labels, the preprocessing pipeline, and the evaluation protocol. The model may perform well on a benchmark and still fail on a different network if traffic distributions change. Consequently, this chapter presents deep learning not as a magic solution, but as a set of tools that must be carefully adapted to the IDS problem. This position is consistent with the general role of IDPS technologies described by NIST, where detection must be understood together with monitoring, configuration, maintenance, and operational response [1].

Figure 2.1 summarizes the problem-driven view adopted in this chapter. Enterprise traffic is high-dimensional, variable, and often ambiguous. Rule-based detection may be efficient for known signatures, but it is less suitable when attacks evolve or when the relevant pattern is distributed across several features. The CNN-LSTM approach used in this thesis is therefore motivated by two complementary needs: learning feature interactions and capturing contextual or sequential behavior.

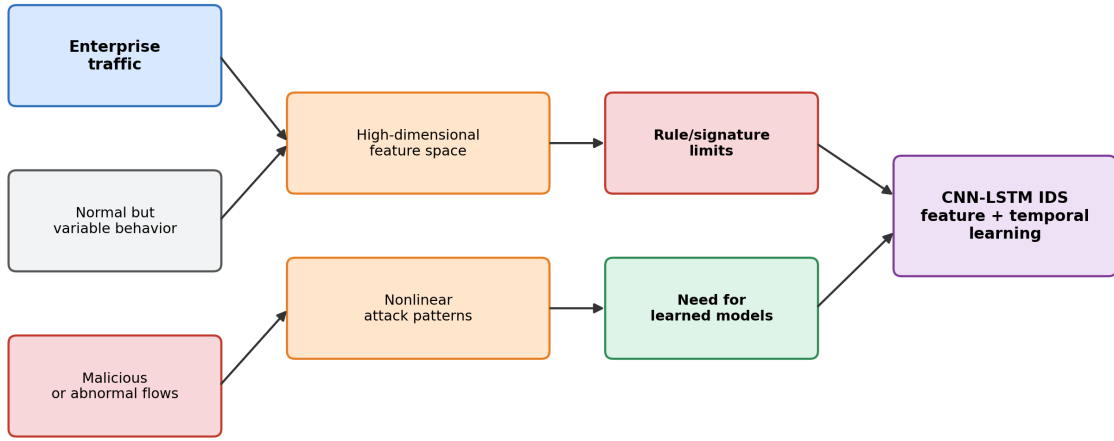


Figure 2.1: Problem-driven motivation for using deep learning in enterprise anomaly detection.

2.2 Nature of Anomalies in Enterprise Network Traffic

An anomaly is generally defined as an observation that deviates from an expected behavior [7]. In a network security context, this definition is useful but incomplete. A deviation is not necessarily malicious. A backup operation, a software update, a remote access session, or a temporary overload may produce traffic patterns that differ from the usual baseline. Conversely, an attacker may deliberately generate traffic that remains close to normal behavior in order to avoid detection. The IDS problem is therefore not only the problem of detecting rare values. It is the problem of interpreting deviations under uncertainty.

In enterprise networks, anomalies appear at different levels. At the packet level, abnormal behavior may involve unusual flags, malformed packets, or unexpected protocol combinations. At the flow level, it may appear as abnormal duration, large byte transfers, repeated connection attempts, or unusual service usage. At the host level, it may be observed as repeated failed logins, access to sensitive services, scanning behavior, or communication with

unknown destinations. At the temporal level, attacks may be visible only through accumulation: a single connection can look innocent, but a sequence of connections may reveal a scan, a brute-force attempt, or a denial-of-service pattern.

This multi-level nature explains why deep learning architectures can be relevant. A feedforward model can classify individual feature vectors, but it may ignore order and context. A convolutional model can learn local interactions between features, but it does not automatically maintain memory. A recurrent model can exploit sequences, but it may be less efficient at discovering local feature combinations from a compact vector. The hybrid CNN–LSTM architecture combines these two principles: convolutional layers first create an informative representation, and the LSTM layer then processes this representation as a contextual signal.

Table 2.1: Practical difficulties that make anomaly detection more complex than a simple binary classification task.

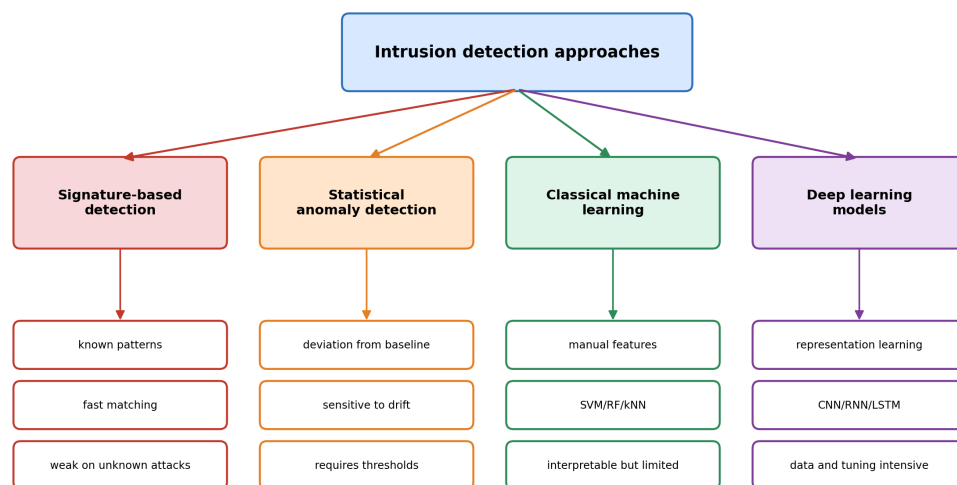
Difficulty	Consequence for IDS design	Implication for the proposed work
Normal traffic variability	The same service can produce different traffic volumes depending on time, user role, and application state.	The model should learn robust patterns instead of relying on one fixed threshold.
High dimensionality	Traffic records contain several numerical and categorical features whose interactions may be nonlinear.	Feature selection and CNN layers are used to reduce and organize the information.
Class imbalance	Some attacks appear less frequently than normal events or common attack classes.	Metrics such as precision, recall, and F1-score are more informative than accuracy alone.
Unknown attacks	Signature-based detection can miss new variants or attacks that were not explicitly encoded.	Representation learning is used to improve generalization beyond manual rules.
Operational constraints	A deployable IDS must run with limited latency and acceptable resource consumption.	The model is kept compact enough for Raspberry Pi deployment.
Dataset limitations	Benchmarks may not perfectly represent current enterprise traffic.	Results are interpreted as validation of the pipeline, not as a universal security guarantee.

2.3 Detection Families: From Rules to Deep Models

Intrusion detection approaches can be grouped into several families. Signature-based detection searches for known patterns. It is often fast and precise when the attack signature is already known, but it is naturally limited against new attacks or modified variants. Statistical anomaly detection defines a baseline and searches for deviations from it. This method can reveal unusual behavior, but it may be sensitive to normal changes in network usage. Classical machine learning methods such as decision trees, support vector machines, random forests, or logistic regression learn a classifier from features. They can be effective, but they often depend strongly on feature engineering and may not fully exploit complex representations.

Deep learning extends machine learning by using multiple layers to learn transformations of the input data [2, 3]. In intrusion detection, this means that the model can learn intermediate representations of traffic records. These representations may combine protocol, service, byte count, connection statistics, and error rates in ways that are difficult to define manually. Nevertheless, deep learning also increases the number of design choices. The architecture, activation functions, loss function, optimizer, threshold, validation method, and feature representation all influence the final behavior.

Figure 2.2 positions deep learning within the broader family of IDS approaches. The proposed work belongs to the anomaly-based and supervised deep learning family. It does not replace all security tools. Instead, it provides an additional detection layer that can complement firewall rules, signatures, and administrator analysis.



The proposed work is positioned in the last family: a supervised deep model that learns feature interactions and sequential dependencies.

Figure 2.2: Taxonomy of intrusion detection approaches and position of deep learning models.

Table 2.2: Comparison of detection strategies from the perspective of enterprise anomaly detection.

Approach	Main idea	Strengths	Main limitations
Signature-based IDS	Match traffic against known attack patterns or rules.	Efficient for known attacks; usually interpretable.	Limited against zero-day attacks and variants.
Statistical anomaly detection	Compare traffic with a normal baseline.	Can reveal abnormal behavior without explicit signatures.	Baseline drift and threshold selection are difficult.
Classical machine learning	Learn a decision boundary from engineered features.	Often efficient and easier to train than deep models.	Feature engineering can dominate performance.
Deep neural networks	Learn hierarchical representations from data.	Can model nonlinear relations and complex combinations of features.	Requires careful validation, tuning, and computational control.
Hybrid CNN–LSTM	Combine local feature extraction with sequence/context modeling.	Suitable for structured traffic features and temporal patterns.	More complex than a single classifier; may overfit if data are limited.

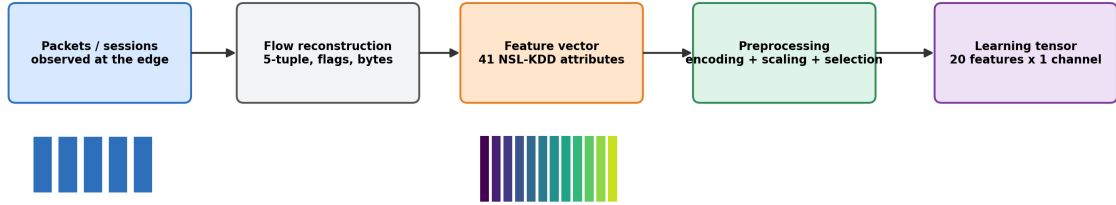
2.4 Transforming Network Traffic into a Learning Problem

A neural network cannot directly understand a network topology, a packet capture, or an administrator’s intuition. It receives numbers. Therefore, the first conceptual step in a learning-based IDS is representation. In packet-based monitoring, raw frames contain low-level information such as addresses, ports, protocols, flags, payload sizes, and timestamps. These raw elements are generally transformed into flows or connection records. A flow representation summarizes communication between endpoints using features such as duration, service, protocol, number of bytes, error rates, and connection statistics.

The NSL–KDD dataset follows this feature-based view. Each record contains 41 input features and a label, which can be converted into a binary normal/attack label [5, 6]. Some features are continuous, such as duration or byte counts; others are categorical, such as protocol type, service, and flag. This heterogeneous structure is typical of IDS datasets. Before training a neural model, categorical features must be encoded, numerical features must be scaled, and irrelevant or redundant dimensions may be removed. In this thesis, the pipeline reduces the initial feature space to 20 informative features before reshaping the data for CNN–LSTM

learning.

Figure 2.3 illustrates this transformation. The figure is important because it shows that the learning model is only one part of the IDS. A poor preprocessing stage can produce misleading results even if the neural architecture is sophisticated. Conversely, a well-controlled preprocessing pipeline can make a compact model more useful for embedded deployment.



The learning model does not process a network directly; it processes a controlled numerical representation of the traffic.

Figure 2.3: Transformation of observed network events into numerical representations used by the learning model.

Table 2.3: Feature representation levels used in network intrusion detection.

Representation level	Examples	Relevance for this thesis
Packet-level information	IP addresses, ports, flags, protocol fields, packet length.	Useful for packet capture systems, but heavier to process directly on small devices.
Flow-level features	Duration, service, byte counts, connection counts, error rates.	Main representation used by NSL-KDD and by the proposed model.
Host-level context	Number of login attempts, compromised files, host-based indicators.	Some NSL-KDD features approximate this type of context.
Window-level representation	Ordered group of recent flow records or feature vectors.	Motivates the use of LSTM when traffic is interpreted as a sequence.
Alert-level information	Model score, predicted class, timestamp, and possible administrator response.	Needed for practical deployment on Raspberry Pi.

2.5 Datasets and the Meaning of Benchmark Evaluation

Benchmark datasets are necessary because they allow researchers to compare models under reproducible conditions. However, a benchmark must not be confused with a complete reproduction of a modern enterprise network. The NSL-KDD dataset was introduced to address important limitations of KDD’99, especially redundancy and biased evaluation [5, 6]. It remains useful for methodological validation and comparison, but it is not a perfect repre-

sensation of current traffic. The Canadian Institute for Cybersecurity states that NSL–KDD still has limitations, even though it can help compare intrusion detection methods [6].

More recent datasets such as CICIDS2017, CSE-CIC-IDS2018, and UNSW–NB15 were designed to provide richer or more modern attack scenarios. CICIDS2017 includes benign traffic and common attacks with PCAP files and extracted flow features [21]. CSE-CIC-IDS2018 includes seven attack scenarios, a larger victim organization, and flow features extracted using CICFlowMeter-V3 [22]. UNSW–NB15 includes nine attack categories and was generated using a cyber range environment with modern normal and synthetic attack behaviors [23]. These datasets are relevant for perspectives, but the present thesis keeps NSL–KDD as the main benchmark because the objective is to validate a complete training-to-deployment pipeline rather than to claim universal superiority.

Table 2.4: IDS datasets cited in the thesis and their methodological role.

Dataset	Main characteristic	Usefulness	Caution
NSL–KDD	Refined version of KDD’99 with reduced redundancy.	Suitable for controlled comparison and for reproducing classical IDS experiments.	Old traffic assumptions; not fully representative of current enterprise networks.
CICIDS2017	Includes benign traffic, common attacks, PCAPs, and extracted CSV flows.	Useful for validating models on richer attack scenarios.	Requires heavier preprocessing and careful class handling.
CSE-CIC-IDS2018	Includes seven attack scenarios and 80 extracted flow features.	Relevant for large-scale modern IDS evaluation.	Dataset size and feature complexity increase computational cost.
UNSW–NB15	Includes nine attack categories and modern normal/synthetic behaviors.	Often used to test generalization beyond KDD-family datasets.	Class overlap and class imbalance can complicate evaluation.

2.6 Supervised Learning Under IDS Constraints

The proposed model is trained in a supervised setting. Each traffic record is associated with a target label indicating whether the record is normal or malicious. In mathematical terms,

the training set can be written as

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N, \quad (2.1)$$

where $x_i \in \mathbb{R}^d$ is a feature vector and $y_i \in \{0, 1\}$ is the binary label. The learning algorithm searches for a function f_θ parameterized by θ such that $f_\theta(x_i)$ approximates the probability that the input belongs to the attack class.

In a classroom example, supervised learning is often presented as a clean mapping from inputs to labels. In intrusion detection, this mapping is less clean. Some labels may be imperfect. Some attacks may share statistical properties with normal traffic. Some normal events may be rare. The model must therefore learn a decision boundary that is robust enough to generalize beyond the exact examples seen during training. This requirement explains why the dataset is divided into training, validation, and test subsets. Training data are used to update weights. Validation data are used to monitor generalization during development. Test data are reserved for final evaluation.

Another IDS-specific difficulty is that the cost of errors is asymmetric. A false positive means that normal traffic is incorrectly flagged as malicious. Too many false positives create alert fatigue and reduce trust in the system. A false negative means that an attack is missed. Too many false negatives make the IDS ineffective. In practice, the preferred balance depends on the deployment context. A critical infrastructure may accept more false positives to avoid missing attacks, while a small office network may prefer fewer alarms to reduce administrator workload. This is why accuracy alone is insufficient. Precision, recall, and F1-score must be interpreted together.

2.7 Artificial Neural Networks as Trainable Decision Functions

An artificial neural network is a composition of simple transformations. A neuron computes a weighted sum of its inputs, adds a bias, and applies a nonlinear activation function. For an input vector $x = [x_1, x_2, \dots, x_d]$, the pre-activation value is

$$z = w^T x + b = \sum_{j=1}^d w_j x_j + b, \quad (2.2)$$

and the neuron output is

$$a = \phi(z), \quad (2.3)$$

where ϕ is an activation function. A neural layer applies several such transformations in parallel. A deep network stacks layers so that the output of one layer becomes the input of

the next one.

For IDS data, this composition is useful because the separation between normal and malicious traffic is rarely linear. For example, a high number of bytes may be normal for a file transfer but suspicious for a service that usually exchanges small messages. A high connection count may be normal for a proxy but suspicious for a workstation. The meaning of a feature depends on other features. Neural networks can learn such interactions by adjusting weights during training.

The training process is summarized in Figure 2.4. During forward propagation, the model produces a prediction. The prediction is compared with the true label through a loss function. During backpropagation, gradients are computed and an optimizer updates the parameters. This cycle is repeated over many mini-batches. The purpose is not to memorize the training data but to learn a function that remains useful on unseen examples [3].

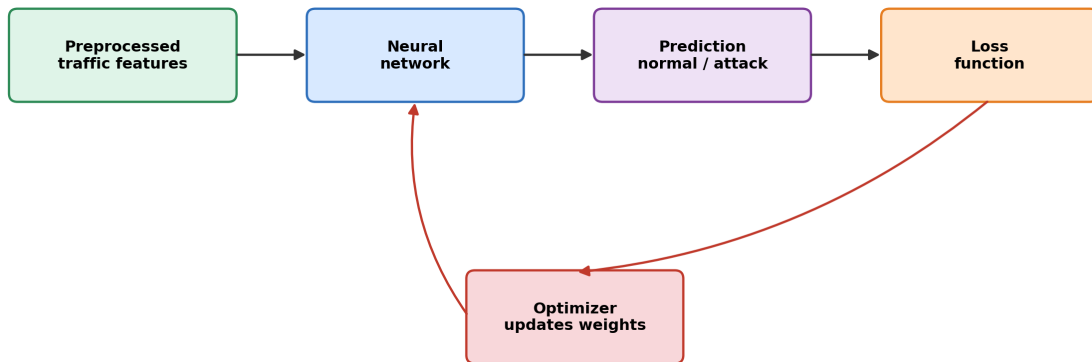


Figure 2.4: Training loop of a supervised neural intrusion detection model.

2.8 Deep Representation Learning for Traffic Features

The main idea of deep representation learning is that the model does not only classify the original input; it also learns intermediate representations that make classification easier. In the first layers, the model may detect simple relations between features. In deeper layers, it may combine these relations into more abstract patterns. In image recognition, this process is often described as going from edges to shapes and then to objects. In intrusion detection, the analogy is not visual, but it remains conceptually useful: the model may go from raw traffic attributes to local feature combinations and then to a global normal/attack decision.

This property is important for NSL-KDD-like data. The dataset includes basic features, content-related features, time-based traffic features, and host-based traffic features. Individually, these features may be weak indicators. Together, they can reveal meaningful behavior. For example, connection counts, same-service rates, and error rates may collectively suggest scanning or denial-of-service behavior. A deep model can learn to combine them without requiring the designer to explicitly write every possible rule.

Nevertheless, representation learning has a risk: if the dataset contains shortcuts or biases, the network may learn them. A model can appear accurate because it exploits dataset artifacts rather than real attack mechanisms. This is why Chapter 3 uses a controlled preprocessing pipeline and Chapter 4 interprets results cautiously. The chapter does not present deep learning as a replacement for security expertise. It presents it as an additional modeling layer whose value depends on data quality and validation.

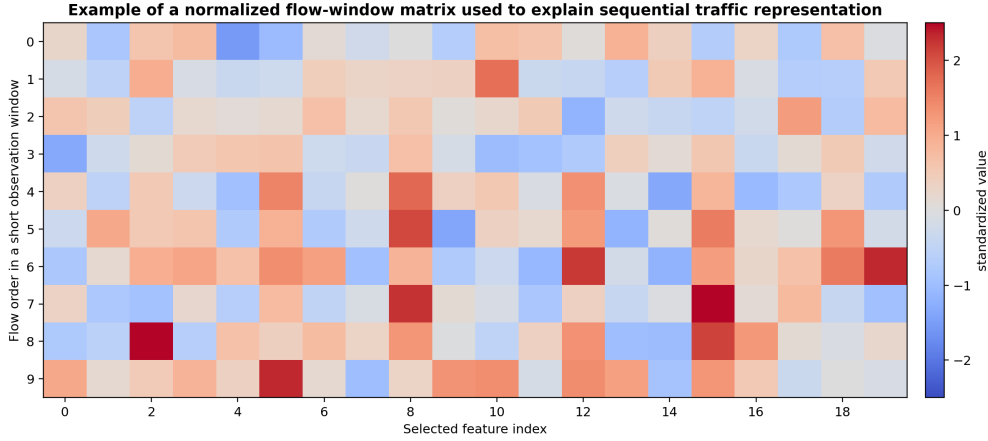


Figure 2.5: Flow-window matrix used to explain how selected traffic features can be organized for sequence-aware learning.

2.9 Convolutional Neural Networks for Feature Interaction Extraction

Convolutional Neural Networks are commonly associated with images, but their principle is more general. A convolutional layer applies trainable filters to local regions of an input. For one-dimensional traffic features, a filter slides across adjacent feature positions and learns combinations that are useful for classification. The order of the features is not naturally spatial as in an image, but after preprocessing and feature selection the vector can still be treated as a structured input for a compact feature extractor.

In the proposed model, Conv1D layers are used before the LSTM layer. This choice has a practical interpretation. The CNN component can emphasize local combinations among selected traffic attributes. These combinations are then passed to the recurrent component. Instead of asking the LSTM to process the raw selected feature vector directly, the model first gives it a transformed representation. This can reduce the burden on the recurrent layer and make the architecture more suitable for a compact embedded implementation.

A one-dimensional convolution can be written as

$$h_t^{(k)} = \phi \left(\sum_{r=0}^{m-1} q_r^{(k)} x_{t+r} + b^{(k)} \right), \quad (2.4)$$

where $q^{(k)}$ is the k -th convolution filter, m is the filter length, and ϕ is a nonlinear activation. The exact interpretation of t depends on how the input vector is arranged, but the operational goal remains the same: learn small patterns that help distinguish normal and attack records. Figure 2.6 illustrates this principle.

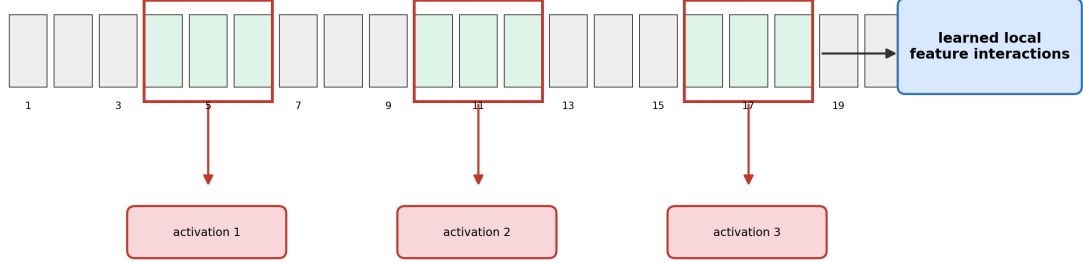


Figure 2.6: One-dimensional convolution used to learn local interactions between selected traffic features.

Pooling layers are often placed after convolutional layers. Their role is to reduce the size of the representation and to retain the most salient activations. In this thesis, max-pooling is used after the convolutional layers. This is consistent with the objective of obtaining a compact representation before the LSTM layer. The pooling operation also helps reduce computational cost, which is useful for Raspberry Pi deployment.

2.10 Recurrent Neural Networks and Temporal Dependence

Network attacks are often temporal phenomena. A port scan is a sequence of connection attempts. A brute-force attack is a sequence of authentication failures. A denial-of-service attack may be visible through a rapid accumulation of similar flows. Even when the dataset is represented as individual records, the underlying security behavior often has a sequential interpretation. This is the main reason for using recurrent models in intrusion detection.

A Recurrent Neural Network processes an input sequence by maintaining a hidden state. At time step t , the hidden state depends on the current input and on the previous hidden state:

$$h_t = \phi(W_x x_t + W_h h_{t-1} + b). \quad (2.5)$$

This recursive form gives the model a memory of previous observations. However, simple recurrent networks are difficult to train on long dependencies because gradients may vanish or explode during backpropagation through time. LSTM networks were introduced to address this problem by using a gated memory mechanism [4].

In an IDS, the value of memory depends on the representation. If the input is a single independent record, a recurrent layer may act mainly as an additional nonlinear transformation.

If the input is a window of flows, the recurrent layer can model how traffic evolves. In both cases, the LSTM idea remains relevant: it provides a controlled mechanism for deciding what information should be preserved, updated, or exposed to the output decision.

2.11 Long Short-Term Memory Units

An LSTM unit contains a cell state and several gates. The forget gate decides what information from the previous cell state should be removed. The input gate decides what new information should be stored. The output gate decides what part of the cell state should influence the hidden output. This structure helps the model maintain useful information over time and reduce the vanishing-gradient problem.

A simplified LSTM formulation can be written as follows:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \quad (2.6)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i), \quad (2.7)$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c), \quad (2.8)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad (2.9)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o), \quad (2.10)$$

$$h_t = o_t \odot \tanh(c_t), \quad (2.11)$$

where f_t , i_t , and o_t denote the forget, input, and output gates, respectively. The operator $\odot$ represents element-wise multiplication. The equations show that the LSTM is not just a deeper feedforward layer; it is a memory mechanism that controls information flow.

Figure 2.7 gives an intuitive view of this mechanism. For intrusion detection, the memory is useful when the classifier must interpret a current observation in relation to what came before. Even when the final implementation uses compact feature tensors, the LSTM layer gives the architecture a capacity to encode contextual dependencies beyond simple feature-level thresholds.

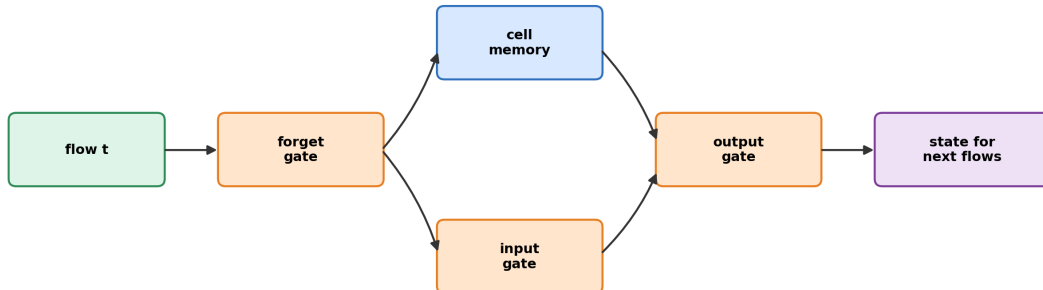


Figure 2.7: LSTM gates as a controlled memory mechanism for traffic sequences.

2.12 Hybrid CNN–LSTM Architecture for the Proposed IDS

The hybrid architecture used in this thesis combines two complementary ideas. The CNN part acts as a feature-interaction extractor. It receives the selected and scaled input representation, applies one-dimensional convolutional filters, and produces feature maps. The LSTM part receives the transformed representation and learns contextual dependencies. The final dense layers map the learned representation to a binary alert score.

This design is motivated by the nature of the IDS problem. Some attack indicators are local combinations of features: for example, a service type combined with abnormal byte counts or error rates. Other indicators are contextual: the significance of a connection may depend on recent traffic behavior. By joining CNN and LSTM layers, the architecture is better aligned with both types of information. Similar motivations appear in the literature on hybrid deep learning for intrusion detection [24, 25, 26].

The implemented architecture follows a compact sequence: input layer, Conv1D layer with 64 filters, max-pooling, Conv1D layer with 128 filters, max-pooling, LSTM layer with 64 units, dense layer with 64 neurons, and a sigmoid output. The model is trained with binary cross-entropy and the Adam optimizer. This architecture is not chosen only for accuracy. It is also chosen because it remains small enough to be exported and loaded on the Raspberry Pi in the deployment stage.



Figure 2.8: Problem-oriented architecture of the CNN–LSTM IDS used in this thesis.

Table 2.5: Functional role of each part of the proposed CNN–LSTM architecture.

Component	Role in the model	Reason for keeping it in the thesis pipeline
Feature selection	Keeps a reduced subset of informative features.	Reduces dimensionality and supports embedded deployment.
Scaling	Normalizes numerical magnitudes.	Prevents large-valued features from dominating training.
Conv1D layers	Learn local interactions among selected features.	Provide a compact representation before the recurrent layer.
Max-pooling layers	Reduce representation size and retain strong activations.	Lower computational cost and help control model complexity.
LSTM layer	Encodes contextual dependencies in the learned representation.	Gives the model memory-like behavior useful for sequence-aware detection.
Dense layers	Convert the learned representation into a decision.	Produce a final binary normal/attack score.
Sigmoid output	Maps the final logit to a probability-like score.	Allows threshold-based alert generation.

2.13 Activation Functions and Their Operational Meaning

Activation functions introduce nonlinearity into neural networks. Without them, a stack of linear layers would still behave like a single linear transformation. In intrusion detection, nonlinearity is essential because the relationship between traffic features and malicious behavior is rarely linear. For example, the same byte count may be normal or suspicious depending on service type, protocol, connection rate, and host behavior.

The Rectified Linear Unit (ReLU) is widely used in hidden layers because it is simple and helps gradients propagate in deep networks [3]. It is defined as

$$\text{ReLU}(z) = \max(0, z). \quad (2.12)$$

When the pre-activation is positive, ReLU passes it unchanged; when it is negative, it outputs zero. This behavior makes hidden representations sparse and computationally efficient.

The sigmoid function is commonly used for binary classification outputs:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \quad (2.13)$$

It maps the final logit to a value between 0 and 1. In this thesis, the sigmoid value is in-

interpreted as an attack score. A threshold is then applied to convert the score into a normal or attack label. This threshold is operationally important. Lowering the threshold usually increases recall but may increase false positives. Raising the threshold usually increases precision but may miss more attacks.

For multiclass IDS tasks, the softmax function is often used. It converts a vector of class scores into a probability distribution over classes. Since the main experimental objective of this thesis is binary classification, sigmoid is more appropriate for the final layer. Multiclass extension remains a perspective for future work.

2.14 Loss Function, Optimizer, and Regularization

The loss function measures the difference between predicted outputs and true labels. For binary classification, binary cross-entropy is a natural choice and is widely implemented in deep learning frameworks such as TensorFlow/Keras [27]. If $y \in \{0, 1\}$ is the true label and $\hat{y}$ is the predicted probability of attack, the binary cross-entropy loss is

$$\mathcal{L}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]. \quad (2.14)$$

This loss penalizes confident wrong predictions strongly. In an IDS, this property is useful because a model should not be confidently wrong about attack traffic.

Optimization is performed using gradient-based methods. The Adam optimizer is commonly used because it adapts learning rates using estimates of first and second moments of the gradients [28]. In the proposed implementation, Adam is used with a learning rate of 0.001. This value provides a reasonable compromise between convergence speed and stability for the selected model.

Regularization is necessary because neural models can overfit. Overfitting occurs when a model performs well on training data but poorly on unseen data. In this thesis, several practical mechanisms reduce this risk: feature selection limits the input dimensionality, validation monitoring is used during training, and early stopping stops training when validation loss no longer improves. These choices are especially important because benchmark datasets may contain artifacts or redundant patterns that a neural network could memorize.

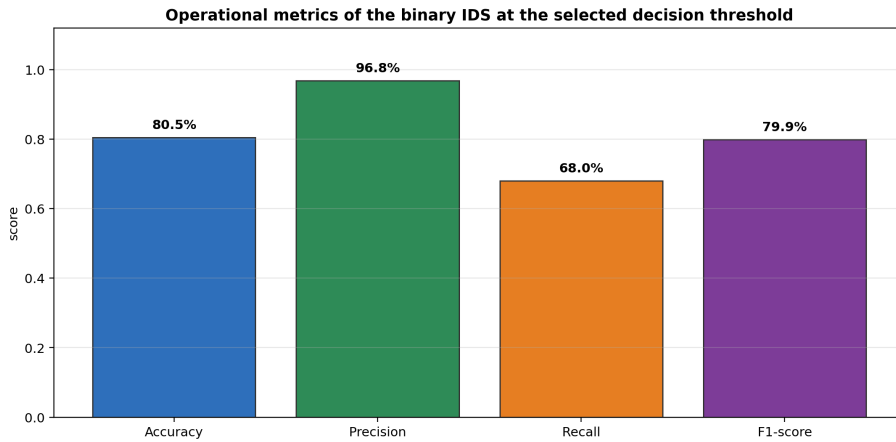
2.15 Decision Threshold and Alert Generation

A binary neural IDS generally produces a continuous score. The score must be converted into an alert decision. The simplest rule is

$$\hat{y} = \begin{cases} 1, & \text{if } f_{\theta}(x) \geq \tau, \\ 0, & \text{otherwise,} \end{cases} \quad (2.15)$$

where τ is the decision threshold. The threshold is not a secondary detail. It directly controls the balance between missed attacks and false alarms. In the experimental pipeline of this thesis, a threshold of 0.30 is used for the binary decision. This threshold favors the detection of more suspicious records while keeping precision high enough for useful alerting.

Figure 2.9 presents the operational metrics reported for the selected threshold. These values are analyzed experimentally in Chapter 4; they are introduced here to explain why model evaluation must be interpreted through the lens of operational use. A precision of 96.8% indicates that most generated alerts correspond to attacks. A recall of 68.0% indicates that some attacks are still missed. This is not a contradiction. It is the typical trade-off of a threshold-based IDS.



Threshold used in the project: 0.30; detailed experimental analysis is reported in Chapter 4.

Figure 2.9: Operational IDS metrics obtained at the selected decision threshold and discussed in Chapter 4.

The confusion matrix is the starting point for IDS evaluation. It contains true positives, true negatives, false positives, and false negatives. In this thesis, the attack class is considered the positive class. A true positive is an attack correctly detected. A true negative is normal traffic correctly accepted. A false positive is normal traffic incorrectly flagged as an attack. A false negative is an attack incorrectly classified as normal. Accuracy is intuitive, but it can be misleading when classes are imbalanced. If most records are normal, a classifier can obtain high accuracy while missing many attacks. For security applications, this situation is

unacceptable. High precision means that when the system raises an alert, the alert is likely to be meaningful. It is useful when a single score is needed, but it should not replace the separate analysis of precision and recall. The table below summarise all the metrics and there meaning for an enterprise IDS.

Table 2.6: Evaluation metrics and their operational interpretation for an IDS.

Metric	Mathematical focus	Operational meaning in an enterprise IDS
Accuracy	Overall proportion of correct decisions.	Useful as a global indicator, but insufficient when attack and normal classes are imbalanced.
Precision	Ratio of correct alerts among all alerts.	Indicates whether the administrator can trust the alarms generated by the IDS.
Recall	Ratio of detected attacks among all attacks.	Indicates the ability of the IDS to avoid missing malicious traffic.
F1-score	Harmonic mean of precision and recall.	Provides a compromise score when precision and recall must be balanced.
False positive rate	Normal samples incorrectly flagged as attacks.	High values create alert fatigue and reduce acceptance of the system.
False negative rate	Attack samples incorrectly accepted as normal.	High values reduce the protective value of the IDS.
Inference latency	Time required to produce a prediction.	Critical for real-time deployment on embedded hardware.

2.16 Model Generalization and Evaluation Pitfalls

A common mistake in intrusion detection research is to treat a high benchmark score as proof that a system is ready for real deployment. This conclusion is too strong. A model can overfit to a dataset. It can also learn artifacts that are specific to the way a dataset was generated. If preprocessing is not performed carefully, information from the test set can leak into training. If the train-test split is not meaningful, the model may be evaluated on examples that are too similar to those used for training.

Another difficulty is concept drift. Network behavior changes over time. New applications are deployed, users change their habits, attackers adapt, and legitimate services evolve. A model trained at one time may become less accurate later. For this reason, a deployable IDS should be considered part of a monitoring pipeline that can be updated, audited, and retrained when necessary. This is particularly important for enterprise environments where

stability and security must be maintained over long periods.

Dataset selection also influences the interpretation of results. NSL-KDD is useful because it is accessible, lightweight, and widely used. It is also limited because it is not a recent capture of a modern production network. CICIDS2017, CSE-CIC-IDS2018, and UNSW-NB15 provide complementary perspectives, but each dataset has its own assumptions and difficulties [19, 21, 22, 23]. A rigorous thesis should therefore avoid exaggerated claims. The present work demonstrates a complete CNN-LSTM IDS pipeline and validates its deployment feasibility; it does not claim to solve intrusion detection for every possible network.

2.17 Embedded Deployment Constraints

The final objective of this thesis is not only to train a model in a notebook. The model is deployed on a Raspberry Pi in a controlled network environment. This deployment objective changes the way deep learning must be discussed. A very large model may obtain better benchmark accuracy but be unsuitable for real-time embedded monitoring. A very small model may be fast but insufficiently expressive. The design must therefore balance detection quality, latency, memory footprint, and preprocessing complexity.

Feature selection plays a central role in this balance. Reducing the feature space from 41 to 20 features lowers the computational load and simplifies deployment artifacts. Scaling and categorical encoders must also be saved and reused exactly during deployment. If the online preprocessing differs from the training preprocessing, the model receives inputs from a different distribution and predictions become unreliable. This is why the project saves the scaler, categorical encoders, selected features, and trained model as deployment artifacts.

Figure 2.10 illustrates the deployment trade-off. In a laboratory environment, it is tempting to focus only on accuracy. In an embedded IDS, the model must also respond within an acceptable time and operate under limited CPU and memory resources. This is one reason why the proposed architecture uses a compact CNN-LSTM rather than a very deep or transformer-based model.

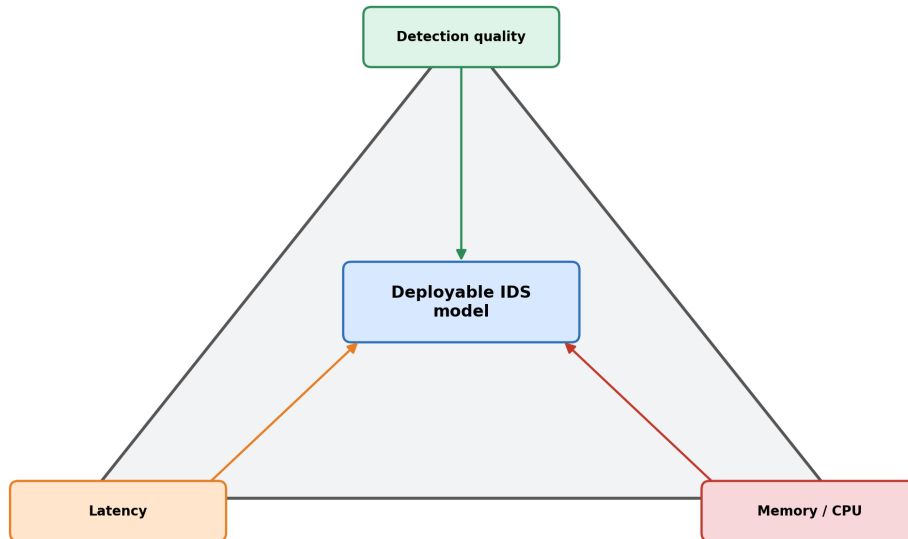


Figure 2.10: Trade-off between detection quality, latency, and resource consumption in embedded IDS deployment.

2.18 Design Choices Retained our work

The theoretical discussion above leads to the design choices used in the rest of the thesis. First, the IDS is formulated as a supervised binary classification problem. This choice is appropriate because NSL-KDD provides labels and because the first operational objective is to distinguish normal traffic from malicious traffic. Second, the preprocessing pipeline is treated as part of the model. Encoding, scaling, feature selection, and tensor reshaping are not secondary details; they determine what the neural network actually sees. Third, the architecture combines CNN and LSTM layers to reflect the dual nature of the problem: local feature interactions and contextual behavior.

Fourth, the evaluation is based on several metrics rather than accuracy alone. The experimental results show high precision and moderate recall. This means that the system is conservative in the sense that its alerts are reliable, but it still misses some attacks. This interpretation is more useful than presenting one score without discussion. Fifth, the deployment objective is considered from the beginning. The model is designed to be exported and executed on Raspberry Pi, which requires compact features and controlled model complexity.

Table 2.7: Synthesis of the Chapter 2 design choices and their relation to the thesis problem.

Design choice	Justification from the problem	Implementation consequence
Binary classification	The first objective is to separate normal traffic from malicious traffic.	Labels are converted into normal/attack classes.
NSL–KDD benchmark	Provides a reproducible labeled dataset for IDS validation.	Training and testing use KDTrain+ and KDDTest+.
Feature selection	Reduces dimensionality and resource consumption.	The feature set is reduced from 41 to 20 selected features.
CNN component	Learns local feature interactions.	Conv1D layers are applied before recurrent processing.
LSTM component	Adds memory-like contextual modeling.	A 64-unit LSTM layer is included after convolution and pooling.
Sigmoid output	Produces a score suitable for binary alerting.	A threshold converts the score into an IDS decision.
Multiple metrics	IDS errors have different operational consequences.	Accuracy, precision, recall, and F1-score are all reported.
Raspberry Pi deployment	The model must be usable in a low-cost monitoring setup.	Preprocessing artifacts and trained model are exported for online execution.

2.19 Conclusion

The chapter therefore prepares the methodology of Chapter 3. It explains why NSL–KDD is used, why the data must be encoded and scaled, why feature selection is necessary, why the model architecture contains convolutional and recurrent layers, and why the performance must be interpreted carefully. This problem-oriented structure strengthens the scientific coherence of the chapter because the discussion is written around the actual project rather than around isolated textbook definitions.

The theoretical elements presented in this chapter can be read as a chain of decisions. The first decision is to model intrusion detection as a learning problem because manual signatures cannot cover all possible behaviors. The second decision is to represent traffic through structured features because neural networks require numerical inputs. The third decision is to reduce the feature space because the project targets an embedded deployment. The fourth decision is to combine CNN and LSTM layers because the security problem involves both feature interactions and contextual patterns. The fifth decision is to evaluate the model through precision, recall, and F1-score because IDS errors have different operational costs.

This chain is more important than any isolated definition. Artificial intelligence, machine learning, deep learning, CNNs, LSTMs, activation functions, and loss functions are not introduced as independent textbook topics. They are introduced because each one answers part of the problem. The CNN is included because selected traffic features can interact. The LSTM is included because network behavior can be sequential. Binary cross-entropy is included because the task is binary classification. The sigmoid output is included because the final decision is threshold-based. Precision and recall are included because an administrator cares about both alert reliability and missed attacks.

This chapter presented deep learning from the perspective of the intrusion detection problem addressed in this thesis. The main idea is that an enterprise IDS must deal with high-dimensional, heterogeneous, variable, and sometimes ambiguous traffic. Signature-based methods remain useful, but they are limited when attacks are unknown or when malicious behavior is distributed across several features. Deep learning provides a way to learn representations from data, provided that the dataset, preprocessing, evaluation, and deployment constraints are handled carefully.

The chapter also explained why the proposed architecture combines CNN and LSTM layers. The CNN part learns interactions among selected traffic features, while the LSTM part introduces memory-like contextual modeling. This hybrid structure is aligned with the nature of network anomalies, where both feature combinations and sequential patterns can be informative. The discussion also emphasized that metrics must be interpreted operationally. Precision, recall, F1-score, threshold selection, and false alarms are directly connected to the usability of the IDS.

Finally, the chapter connected theory to the practical implementation. The model is not developed only for offline classification. It is prepared for Raspberry Pi deployment, which requires a compact input representation, exported preprocessing artifacts, and controlled computational cost. The next chapter uses this foundation to describe the NSL-KDD dataset, the preprocessing pipeline, the selected features, the CNN-LSTM architecture, and the training configuration used in the experimental part of the thesis.

Chapter 3

Datasets and Methodology

3.1 Introduction

This chapter presents the dataset and methodology used to build the proposed hybrid intrusion detection system. The objective is to transform raw traffic records into a clean and consistent numerical representation that can be processed by a CNN–LSTM model. The methodology includes dataset selection, preprocessing, categorical encoding, scaling, feature selection, model design, training configuration, and evaluation metrics.

This chapter operationalizes the design choices established in Chapter 2: the binary normal/attack formulation, the use of NSL–KDD, the 41-to-20 feature reduction, the CNN–LSTM architecture, the threshold-based decision rule, and the need for reproducible preprocessing artifacts.

The experimental work is based on the NSL–KDD dataset. Although this benchmark does not fully represent all characteristics of current enterprise traffic, it remains useful for controlled comparison because it provides labeled records, known attack categories, and a standardized train/test split. The limitations of this dataset are explicitly considered in the discussion and future perspectives.

3.2 NSL–KDD Dataset

The NSL–KDD dataset was proposed as an improved version of the KDD Cup 99 dataset. The original KDD Cup 99 benchmark has been widely used in intrusion detection research, but it contains a large number of redundant records. This redundancy can bias learning algorithms because frequent records may dominate training and evaluation. Tavallaee et al. analyzed these limitations and proposed NSL–KDD to reduce the effect of duplicated and redundant instances [5]. The official dataset description also indicates that redundant records were removed from the training set and duplicate records were removed from the test set [6].

Each NSL–KDD record describes a network connection using 41 features and one label. These features include basic connection information, content-based attributes, and traffic-based statistics. The labels can be used for multiclass attack classification or converted into binary labels for distinguishing normal traffic from attack traffic. In this thesis, the binary formulation is adopted:

- **Normal traffic:** legitimate connection records;
- **Attack traffic:** all malicious records grouped into one attack class.

The original attack types are usually grouped into four main categories:

- **Denial of Service (DoS):** attacks that attempt to make a service unavailable;
- **Probe:** surveillance and scanning activities used to collect information;
- **Remote to Local (R2L):** attempts to gain local access from a remote machine;
- **User to Root (U2R):** attempts by a local user to obtain root privileges.

Table 3.1 summarizes the main characteristics of the dataset.

Table 3.1: Description of the NSL–KDD dataset used in this thesis.

Parameter	Description
Dataset name	NSL–KDD
Dataset type	Network-based intrusion detection benchmark
Training records	KDDTrain+: 125,973 records
Testing records	KDDTest+: 22,544 records
Number of features	41 traffic and connection features
Attack categories	DoS, Probe, R2L, and U2R
Binary formulation	Normal traffic vs. attack traffic
Main improvement over KDD Cup 99	Reduction of redundant records in the training set and removal of duplicate records from the test set [5, 6]
Feature families	Basic features, content features, and traffic features
Use in this thesis	Training, validation, and testing of the proposed CNN–LSTM IDS

3.3 Preprocessing Pipeline

Preprocessing is a critical step in intrusion detection because neural networks require numerical and consistent input data. In this thesis, the preprocessing pipeline was designed to

ensure that the same transformations used during offline training can also be reused during Raspberry Pi deployment. This consistency is necessary: if the online system encodes or scales features differently from the training phase, the model predictions become unreliable. The adopted preprocessing pipeline includes the following stages:

1. loading the NSL–KDD training and test files;
2. assigning feature names and identifying categorical variables;
3. converting labels into binary classes;
4. checking missing values and duplicate records;
5. encoding categorical attributes;
6. scaling numerical features;
7. selecting the most informative features;
8. reshaping the data for the CNN–LSTM architecture.

Figure 3.1 summarizes this chain.

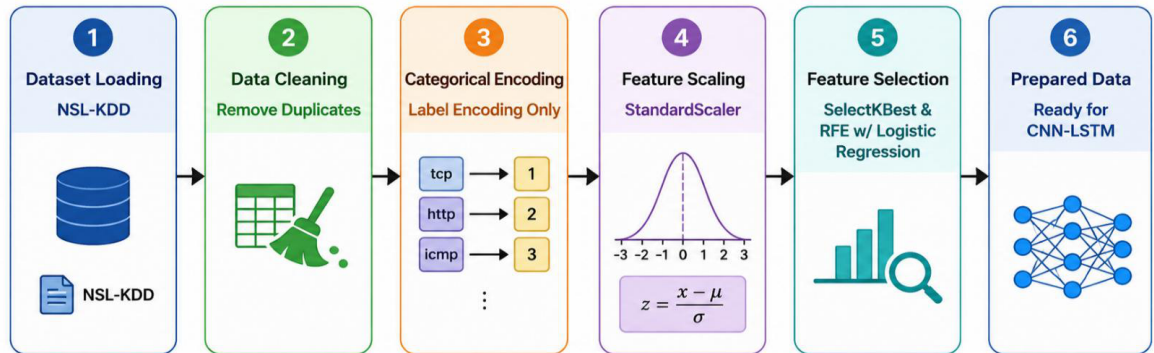


Figure 3.1: Preprocessing pipeline used before model training and deployment.

3.3.1 Data Cleaning and Label Conversion

The first step consists in loading the train and test files and assigning the 41 feature names. The original labels contain several attack names. For binary classification, all attack names are mapped to the attack class, while the normal label is kept as normal. This transformation is consistent with the objective of this thesis, which is to detect whether traffic is normal or malicious.

Cleaning also includes checking for missing values and duplicated records. Even though NSL–KDD was designed to reduce redundancy compared with KDD Cup 99, verifying data consistency remains necessary before training. This step also improves reproducibility because all subsequent transformations are applied to a clearly defined dataset.

3.3.2 Categorical Encoding

Some NSL–KDD features are categorical, such as protocol type, service, and flag. Neural networks cannot process these symbolic values directly. They must therefore be converted into numerical form. In this work, categorical encoders were fitted on the training data and then applied to the validation, test, and deployment data. This avoids changing category-to-number mappings between training and inference.

3.3.3 Feature Scaling and selection

The numerical features have different ranges and units. For example, byte counts and binary indicators do not have the same scale. If these values are used directly, features with large numerical magnitudes may dominate the learning process. Therefore, standardization was applied to center and scale the selected numerical variables. The same scaler was saved and reused during deployment. In contrast, feature selection reduces the dimensionality of the input while preserving the most useful information. It can improve training efficiency, reduce overfitting risk, and facilitate embedded deployment. In this thesis, a two-stage strategy was adopted. First, an ANOVA-based SelectKBest method was used to retain the highest-scoring features. Second, Recursive Feature Elimination (RFE) was applied to refine the feature subset. Scikit-learn describes univariate feature selection as a preprocessing step that selects features according to statistical tests, and SelectKBest keeps the k highest-scoring features [29, 30]. RFE is a wrapper method that recursively removes less relevant features according to an estimator.

The final feature set contains 20 selected features instead of the original 41. This reduction is important for Raspberry Pi deployment because it decreases the amount of data processed by the online inference pipeline.

3.4 Proposed CNN–LSTM Hybrid Model

The proposed model combines convolutional and recurrent processing. The CNN part extracts local feature patterns from the preprocessed traffic representation, while the LSTM part captures dependencies across the learned representation. This design follows the general motivation of hybrid CNN–LSTM models in intrusion detection, where CNN layers are used for representation learning and LSTM layers are used for sequential modeling [20, 24, 25, 26].

3.4.1 CNN Component

The CNN component uses one-dimensional convolutional layers. These layers apply trainable filters to the input representation in order to learn local combinations of features that are useful for classification. Max-pooling reduces the dimensionality of the representation and keeps the most salient activations. ReLU activation is used in the hidden layers to introduce nonlinearity.

3.4.2 LSTM Component

The LSTM component receives the representation produced by the convolutional stage. Its role is to model dependencies that may not be captured by isolated feature inspection. In intrusion detection, this is useful because some attacks are characterized by patterns distributed over several observations or feature groups. The LSTM unit is appropriate for this purpose because it was designed to preserve relevant information over longer sequences than standard recurrent networks [4].

3.4.3 Dense Output Layer

The final dense layer performs binary classification. A sigmoid activation function produces a value between 0 and 1, interpreted as the estimated probability that the analyzed traffic corresponds to an attack. The output decision is defined as:

- 0 $\rightarrow$ normal traffic;
- 1 $\rightarrow$ attack traffic.

The training objective is binary cross-entropy:

$$\text{Loss} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)], \quad (3.1)$$

where y_i is the true binary label and p_i is the predicted attack probability. This loss is suitable for binary classification problems and is implemented in TensorFlow/Keras [27].

Figure 3.2 presents the proposed architecture.

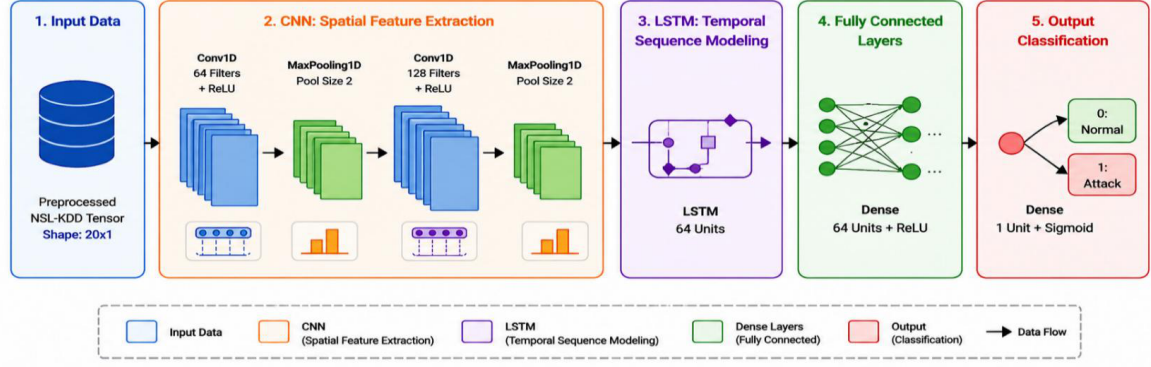


Figure 3.2: Proposed CNN-LSTM hybrid IDS architecture.

3.5 Model Training Configuration

The model was implemented with TensorFlow and Keras. The Adam optimizer was selected because it adapts parameter updates using estimates of the first and second moments of the gradients [28, 31]. The learning rate was set to 0.001. The general update form can be written as:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}, \quad (3.2)$$

where w_t is the updated parameter vector, η is the learning rate, $\hat{m}_t$ is the bias-corrected first-moment estimate, $\hat{v}_t$ is the bias-corrected second-moment estimate, and ϵ is a small numerical stability constant.

To reduce overfitting, early stopping was used with a patience of five epochs. The validation loss was monitored, and the best weights were restored when training stopped. In addition, 20% of the training data was used for validation. Because missing attacks is more critical than generating a limited number of false alarms, class weighting was introduced to increase the importance of attack samples during training.

Finally, the classification threshold was set to 0.3 rather than the default value of 0.5. This design choice increases sensitivity to attacks, but it may also increase false positives. The threshold is therefore a security-policy parameter rather than a universal constant. Table 3.2 summarizes the model configuration.

Table 3.2: Summary of model hyperparameters and training configuration.

Parameter	Value
Conv1D Layer 1 – Filters / Kernel Size	64 / 3
Conv1D Layer 2 – Filters / Kernel Size	128 / 3
MaxPooling1D Pool Size	2
LSTM Units	64
Dense Units	64
Learning Rate	0.001
Batch Size	64
Epochs	30
Validation Split	20%
Early Stopping Patience	5

3.6 Evaluation Metrics

The model is evaluated using a confusion matrix and four derived metrics: accuracy, precision, recall, and F1-score. These indicators are necessary because intrusion detection involves a trade-off between missing attacks and generating false alarms.

3.6.1 Confusion Matrix

The confusion matrix separates predictions into four categories:

- **True Positive (TP):** an attack correctly classified as an attack;
- **True Negative (TN):** normal traffic correctly classified as normal;
- **False Positive (FP):** normal traffic incorrectly classified as an attack;
- **False Negative (FN):** an attack incorrectly classified as normal.

False negatives are particularly dangerous in security systems because they correspond to missed attacks. False positives are also problematic because they increase alert fatigue and reduce administrator confidence.

3.6.2 Accuracy

Accuracy measures the proportion of correctly classified samples:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \quad (3.3)$$

Although accuracy is useful, it should not be interpreted alone when classes are imbalanced.

3.6.3 Precision

Precision measures how many predicted attacks are truly attacks:

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (3.4)$$

High precision means that generated alerts are reliable.

3.6.4 Recall

Recall measures how many real attacks are detected:

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (3.5)$$

High recall means that the system misses fewer attacks.

3.6.5 F1-score

The F1-score is the harmonic mean of precision and recall:

$$F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (3.6)$$

It provides a balanced view when both false positives and false negatives are important.

3.7 Feature Selection as a Security and Deployment Step

Feature selection is sometimes presented as a purely technical preprocessing operation. In this thesis, it has a broader meaning. It is both a statistical step and a deployment-oriented security design step. Statistically, it aims to remove weak, redundant, or noisy variables. Operationally, it reduces the amount of information that must be processed online. This matters because the Raspberry Pi deployment requires the same decisions as the training environment, but with less computational comfort than a cloud notebook or a desktop computer.

The initial NSL–KDD feature vector contains 41 attributes. Keeping all features may appear safer because no information is discarded. However, a larger feature space can also increase noise, redundancy, training instability, and inference cost. Some features may be weakly informative for the binary task. Others may be correlated with more useful features. A compact feature set can improve the practical behavior of the system, especially when the objective is to load preprocessing artifacts and run inference repeatedly in a live environment. The implemented pipeline uses a two-stage selection strategy. First, a filter method based on the ANOVA F-test ranks features according to their relationship with the class label. Second, Recursive Feature Elimination (RFE) with a logistic model is used to retain a final compact subset. The objective is not to claim that these 20 features are universally optimal for every IDS dataset. The objective is to obtain a stable, reproducible, and deployable representation for the CNN–LSTM model used in this work. Figure 3.3 summarizes this interpretation.

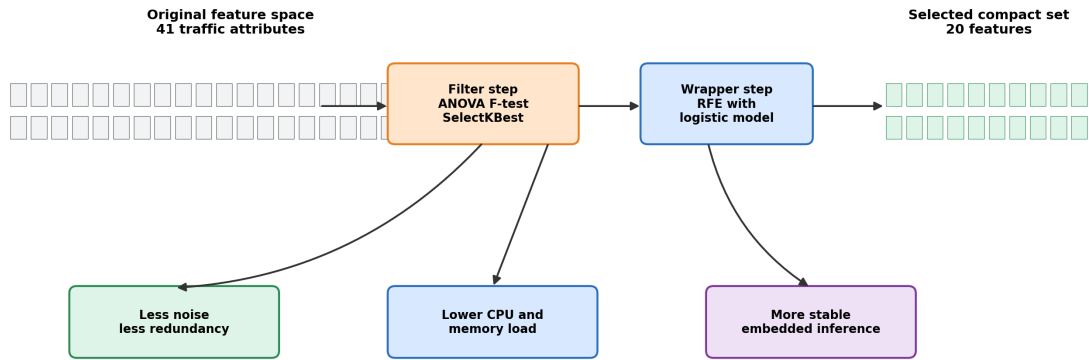


Figure 3.3: Feature selection interpreted as both a learning step and an embedded deployment step.

This design choice also strengthens the methodological coherence of the thesis because the discussion is directly connected to the implemented project. Rather than presenting feature selection as an isolated preprocessing concept, the thesis explains why it is needed for this specific CNN–LSTM IDS and why it matters for Raspberry Pi execution. The objective is to make the proposed chain easier to reproduce, debug, and deploy while maintaining consistency between offline training and online inference.

Table 3.3: Feature-selection choices and their impact on the proposed IDS.

Choice	Technical effect	Practical effect in this thesis
ANOVA F-test	Scores each feature according to its discriminative relation with the class label.	Provides a transparent first reduction before wrapper selection.
Recursive Feature Elimination	Iteratively removes less useful variables according to an estimator.	Produces a compact 20-feature representation for the CNN–LSTM input.
Saved feature list	Ensures that the same variables are used during training and deployment.	Prevents online inference from using inconsistent feature ordering.
Reduced input dimension	Decreases the amount of data processed by the model.	Helps Raspberry Pi deployment by reducing memory and inference burden.
Reproducible preprocessing	Stores the scaler, encoders, selector, and selected feature names.	Makes the experimental pipeline easier to verify and repeat.

3.8 Interpretability, Human Supervision, and Trust

A deep learning IDS should not be considered an autonomous replacement for the network administrator. The system produces alerts, but the meaning of those alerts must still be interpreted. An alert can indicate a real attack, a misconfiguration, a rare legitimate event, or an artifact of the dataset used for training. For this reason, trust in an IDS depends not only on model accuracy but also on the clarity of the pipeline, the stability of preprocessing, and the ability to inspect decisions.

The high precision reported in the project is valuable because it means that most generated alerts are expected to be meaningful. However, the recall remains moderate, which means that the system does not detect all malicious records. A responsible interpretation must therefore avoid exaggerated claims. The model can support monitoring, but it cannot guarantee complete protection. It should be combined with other security mechanisms such as firewall rules, system logs, authentication monitoring, and administrator review.

Human supervision is also important for model maintenance. Network behavior changes, and the system may need to be updated. Logs of suspicious flows and model decisions can help identify false positives and false negatives. These observations can later support retraining, threshold adjustment, or feature engineering. Figure 3.4 shows this practical view. The model is part of a monitoring loop. The loop begins with traffic capture and preprocessing, continues with model scoring and alert generation, and ends with human analysis and possible model

improvement.

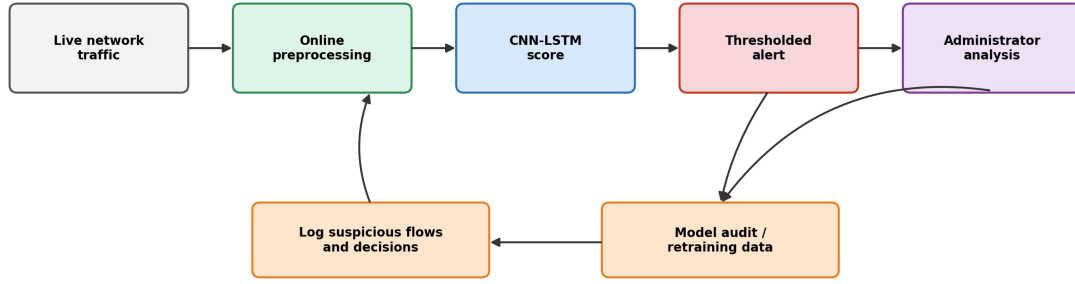


Figure 3.4: Monitoring loop showing how deep learning alerts should remain connected to administrator analysis.

This human-centered view is particularly important in academic projects because benchmark evaluation can create a false sense of completeness. A thesis model may perform correctly on KDDTest+, but real networks include encryption, changing services, new protocols, and user behaviors that are not necessarily represented in the benchmark. Therefore, the role of the proposed system should be formulated precisely: it is a low-cost, deployable prototype that demonstrates the feasibility of CNN–LSTM based anomaly detection on embedded hardware. It is not a certified commercial IDS and it does not remove the need for expert supervision.

3.9 Methodological Limits

The adopted methodology has several limits that must be acknowledged. First, NSL–KDD is an older benchmark and cannot fully reproduce modern enterprise traffic, encrypted flows, cloud applications, or recent attack scenarios. More recent datasets such as CICIDS2017, CSE-CIC-IDS2018, and UNSW–NB15 include more contemporary traffic conditions and should be considered in future evaluation [21, 22, 23]. Second, the binary classification formulation simplifies the problem by grouping all attacks into a single class. A multiclass approach would provide more detailed information to administrators. Third, online deployment requires strict consistency between the training preprocessing pipeline and the real-time inference pipeline.

3.10 Conclusion

This chapter described the dataset and methodology used to develop the proposed IDS. The NSL–KDD dataset was selected because it provides labeled traffic records and a recognized benchmark for intrusion detection research. The preprocessing pipeline converted the raw records into numerical inputs through cleaning, encoding, scaling, and feature selection.

A hybrid CNN–LSTM model was then defined for binary classification between normal and attack traffic. The training configuration used Adam optimization, binary cross-entropy, early stopping, validation monitoring, class weighting, and a tunable decision threshold. The next chapter presents the implementation environment, experimental results, Raspberry Pi deployment, and critical analysis of the proposed system.

Chapter 4

Implementation and Results

4.1 Introduction

This chapter presents the implementation, evaluation, and deployment results of the proposed CNN–LSTM-based intrusion detection system. The previous chapter described the dataset preparation, preprocessing strategy, model architecture, and training procedure. The present chapter focuses on the practical outcome of this implementation, including the offline evaluation on the NSL-KDD test set and the real-time deployment of the trained model on a Raspberry Pi-based experimental platform.

The evaluation follows two complementary directions. First, the trained model is assessed using standard classification metrics, the confusion matrix, and the training/validation curves obtained during the offline learning phase. Second, the trained model and its associated preprocessing artifacts are transferred to the Raspberry Pi, where live network traffic is captured, converted into NSL-KDD-style features, preprocessed, and classified in real time. This organisation makes it possible to evaluate not only the classification capacity of the model, but also the feasibility of deploying the complete IDS chain on a low-cost embedded platform.

4.2 Experimental Environment

4.2.1 Hardware Components

The experimental platform was built to reproduce a small local area network in which normal and abnormal traffic could be generated, captured, and analyzed under controlled conditions. The testbed included a Cisco RV130 router, a TP-Link TL-SG108PE managed switch, a Raspberry Pi 4, an Arduino Ethernet board, and two laptop computers.

The Cisco RV130 router acted as the network gateway. The TP-Link TL-SG108PE switch provided wired connectivity between all devices and was configured to mirror traffic toward

the Raspberry Pi. The Raspberry Pi 4 was used as the embedded deployment platform for the trained IDS. It captured traffic from the mirrored switch port, extracted the required flow-based features, applied the same preprocessing logic used during training, and performed real-time classification.

The Arduino Ethernet board was used as a controlled traffic generator. In the implemented code, the Arduino repeatedly sends HTTP GET requests to a target host in the local network. Therefore, its role is to create reproducible abnormal or attack-like traffic for laboratory validation. Since the traffic originates from a single source node, it should not be interpreted as a true distributed denial-of-service attack.

Two laptops were also included in the platform. One laptop acted as a monitoring station used to observe terminal outputs, alerts, and log files generated by the IDS. The other laptop represented a legitimate client device and generated normal traffic inside the local network.

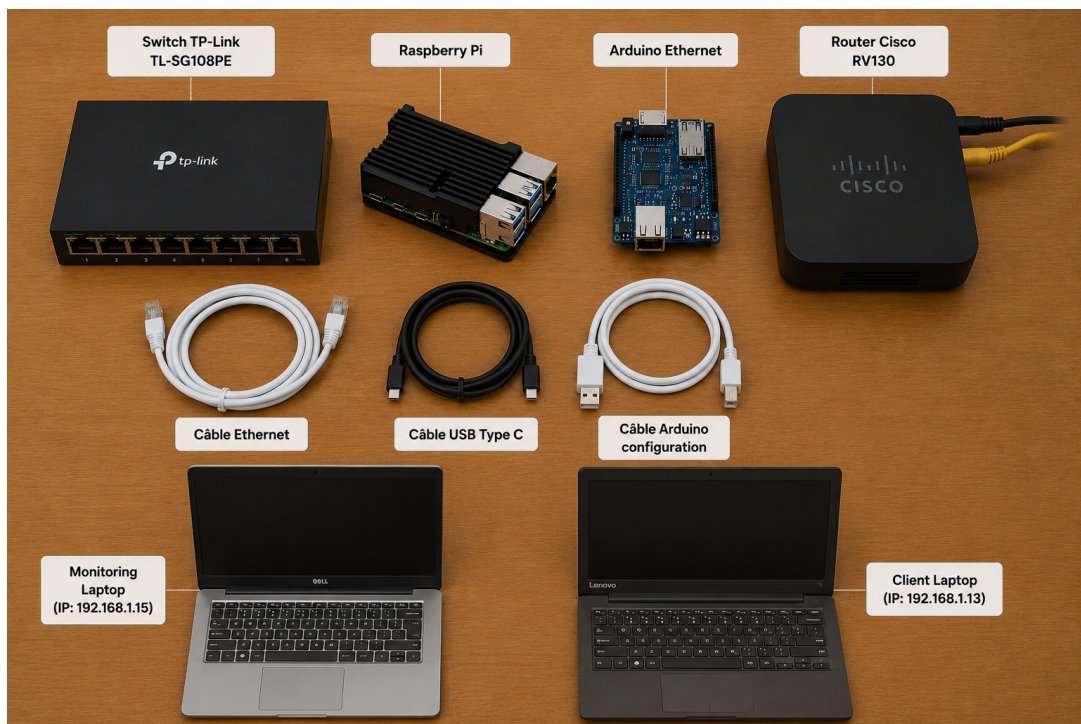


Figure 4.1: Hardware components used to implement the experimental platform.

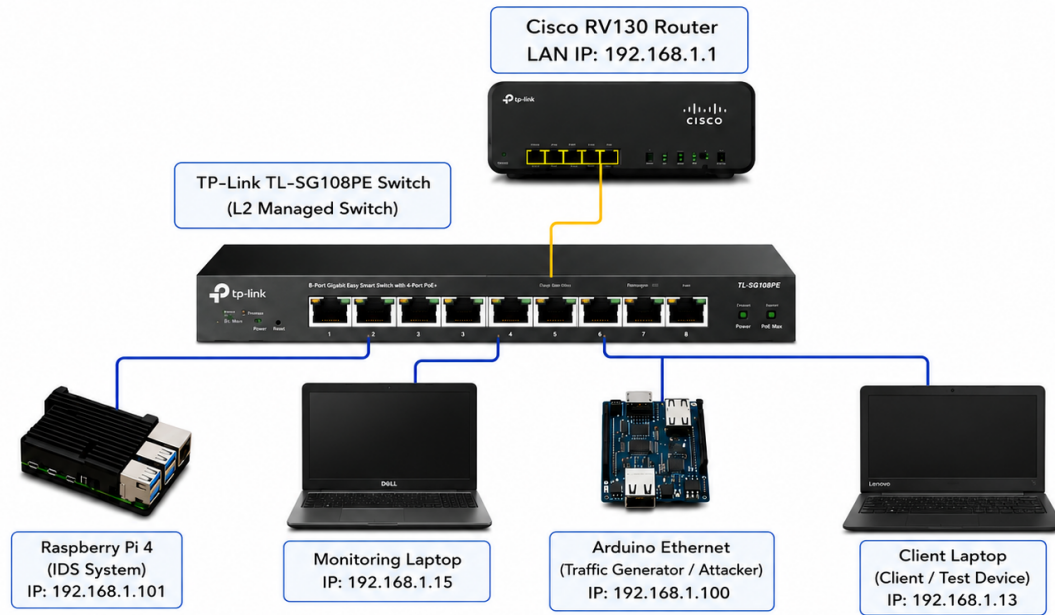
4.2.2 Network Topology

The devices were interconnected to form a local experimental network. The Cisco RV130 router was configured with the address 192.168.1.1 and served as the gateway. The TP-Link managed switch acted as the central communication point and connected the Raspberry Pi, the Arduino Ethernet board, the monitoring laptop, and the client laptop.

To enable passive monitoring, the switch was configured with port mirroring. A copy of the traffic crossing the selected ports was forwarded to the Raspberry Pi, allowing the IDS to

inspect packets without modifying or interrupting normal communications. This configuration is suitable for IDS deployment because the monitoring device observes traffic passively rather than acting as an inline gateway.

In the implemented testbed, the Raspberry Pi used the address 192.168.1.101, the monitoring laptop used the address 192.168.1.15, the client laptop used the address 192.168.1.13, and the Arduino Ethernet board used the address 192.168.1.100. Figure 4.2 illustrates the resulting topology.



IP Addressing Plan				
Device	Role	IP Address	Subnet Mask	Gateway
Cisco RV130 Router (LAN)	Gateway / Router	192.168.1.1	255.255.255.0	-
Raspberry Pi 4 (IDS)	IDS / Monitor	192.168.1.101	255.255.255.0	192.168.1.1
Monitoring Laptop	Monitoring / Management	192.168.1.15	255.255.255.0	192.168.1.1
Arduino Ethernet (Attacker)	Traffic Generator / Attacker	192.168.1.100	255.255.255.0	192.168.1.1
Client Laptop	Client / Test Device	192.168.1.13	255.255.255.0	192.168.1.1

Figure 4.2: Network topology used for the deployment and evaluation of the proposed intrusion detection system.

4.2.3 Software Environment

The software implementation was divided into two main phases: an offline training phase and an online deployment phase. The CNN-LSTM model was developed and trained in Google Colab using Python, TensorFlow/Keras, and Scikit-learn. Google Colab was used because it provides a convenient environment for dataset processing, model training, and GPU-accelerated experimentation.

After training, the model and the preprocessing objects required during deployment were ex-

ported. These artifacts were then transferred to the Raspberry Pi. During the online phase, the Raspberry Pi used Python and Scapy to capture packets from the mirrored switch port. The captured packets were grouped into flows, converted into NSL-KDD-style features, pre-processed using the saved training artifacts, and then classified by the trained CNN-LSTM model.

Table 4.1: Software tools and frameworks used in the proposed IDS implementation.

No.	Software / Tool	Purpose / Description	Role in the Project
1	Python	High-level programming language.	Used for dataset processing, model training, evaluation, and Raspberry Pi deployment.
2	TensorFlow	Deep learning framework.	Used as the computational backend for training and inference.
3	Keras	High-level deep learning API.	Used to define, train, save, and load the CNN-LSTM model.
4	Scikit-learn	Machine learning and preprocessing library.	Used for encoding, scaling, feature selection, evaluation metrics, and artifact export.
5	Google Colab	Cloud-based notebook environment.	Used for the offline training and evaluation phase.
6	Scapy	Packet capture and manipulation library.	Used on the Raspberry Pi to sniff live packets from the mirrored port.
7	Raspberry Pi OS	Operating system for Raspberry Pi.	Used as the execution environment for the online IDS script.
8	Visual Studio Code	Code editor.	Used to write and organize the project source files.
9	Arduino IDE	Arduino programming environment.	Used to program the Arduino Ethernet traffic generator.

4.2.4 Developed Software Modules

The implementation relied on several project-specific files. The notebook `Intrusion_Detection(2).ipynb` was used for exploratory experiments and preliminary comparisons between different intrusion detection approaches. The notebook `model_cnn_lstm_finalll.ipynb` was retained as the final training notebook. It implements the complete offline pipeline: loading the NSL-KDD data, cleaning and encoding the features, standardizing numerical variables, selecting relevant features, training the CNN-LSTM model, evaluating it, and exporting the artifacts required for deployment.

For the online phase, two Python scripts were considered. The first script, `app.py`, corresponds to an initial real-time IDS prototype. It captures live traffic and computes a reduced set of NSL-KDD-compatible features for classification. The second script, `test_ids.py`, represents the improved deployment version. It loads the saved model and preprocessing artifacts, reconstructs flow-based statistics from live packets, applies the same preprocessing chain used during training, reshapes the input according to the CNN-LSTM model requirements, performs inference, and displays or logs the resulting decision.

The Arduino file `sketch_finna1.ino` generates repeated HTTP requests inside the local network. This code is used to produce controlled traffic for validating the responsiveness of the deployed IDS under abnormal traffic conditions.

Table 4.2: Main developed software files and their roles in the proposed IDS workflow.

File / Module	Main Function	Role in the Proposed System
<code>Intrusion_Detection(2).ipynb</code>	Exploratory notebook	Preliminary experiments and comparison of several IDS approaches on NSL-KDD.
<code>model_cnn_lstm_finalll.ipynb</code>	Final training notebook	Complete preprocessing, feature selection, CNN-LSTM training, evaluation, and artifact export.
<code>app.py</code>	Initial online IDS script	First real-time packet capture and feature extraction prototype.
<code>test_ids.py</code>	Improved online IDS script	Flow tracking, reuse of saved preprocessing artifacts, real-time inference, alert display, and CSV logging.
<code>sketch_finna1.ino</code>	Arduino traffic generator	Generation of repeated HTTP request traffic for controlled live testing.

4.2.5 Integrated Software Pipeline

Figure 4.3 summarizes the complete software pipeline. The process starts with the NSL-KDD dataset, continues with preprocessing and CNN-LSTM model training, and ends with the deployment of the saved model on the Raspberry Pi. The same preprocessing objects learned during the offline stage are reused during the online stage. This is an important implementation point because the model can only produce meaningful predictions if the live feature vector is transformed in the same way as the training data.

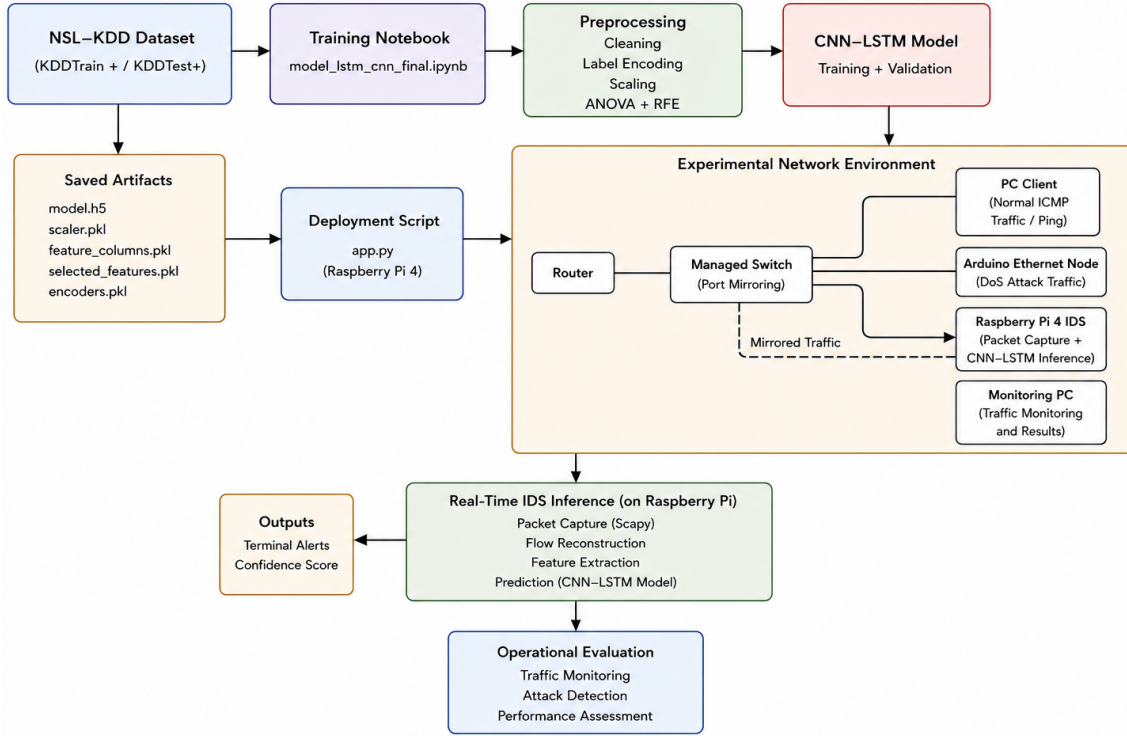


Figure 4.3: End-to-end software pipeline of the proposed IDS, from dataset preparation to online deployment.

The exported artifacts include the trained CNN-LSTM model, the feature lists, the categorical encoders, the scaler or preprocessing pipeline, the selected feature subset, and the label encoder. During deployment, these files are loaded before packet capture starts. The online script then applies the same transformations as the training notebook before submitting the final feature vector to the model.

4.2.6 Roles of the Main Software Modules

Figure 4.4 illustrates the separation between the main software modules. The exploratory notebook is used for experimentation, the final notebook generates the trained model and preprocessing artifacts, the Arduino code generates controlled network traffic, and the Raspberry Pi script executes the real-time IDS. This modular organisation improves reproducibil-

ity because each file has a clearly defined role in the complete workflow.

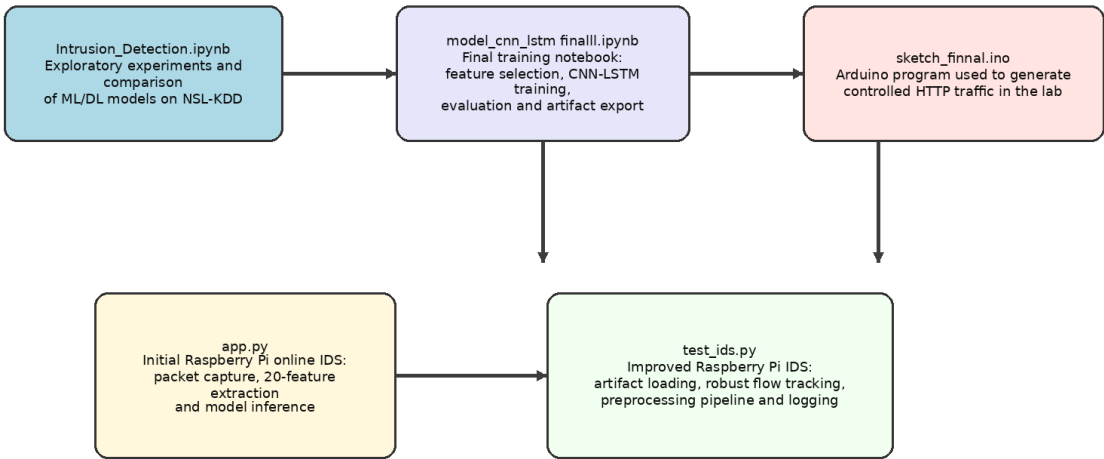


Figure 4.4: Roles of the main software modules used throughout the project.

4.3 Classification Results

4.3.1 Overall Performance Metrics

The trained CNN–LSTM model was evaluated on the NSL-KDD test set using the metrics introduced in the previous chapter. Table 4.3 summarizes the main classification results obtained during the offline evaluation phase.

Table 4.3: Summary of classification performance metrics on the NSL-KDD test set.

Metric	Value	Interpretation
Accuracy	80.5%	Percentage of correctly classified test samples.
Precision	96.8%	Reliability of generated attack alerts.
Recall	68.0%	Proportion of attacks correctly detected.
F1-score	79.9%	Balance between precision and recall.
False positives	287	Normal samples incorrectly classified as attacks.

The results show that the proposed CNN–LSTM model provides a high precision of 96.8%. This means that most alerts generated by the system correspond to attack samples, which is important for practical intrusion detection because a high number of false alarms can reduce administrator confidence in the system.

The recall value of 68.0% indicates that the model does not detect all attack samples. Therefore, the model favors alert reliability more than maximum attack coverage. This behavior can be useful in a monitoring context where false alarms must be limited, but it also shows

that further improvements are possible, especially for attack classes that are difficult to distinguish from normal traffic.

4.3.2 Confusion Matrix Analysis

Figure 4.5 presents the confusion matrix obtained on the NSL-KDD test set.

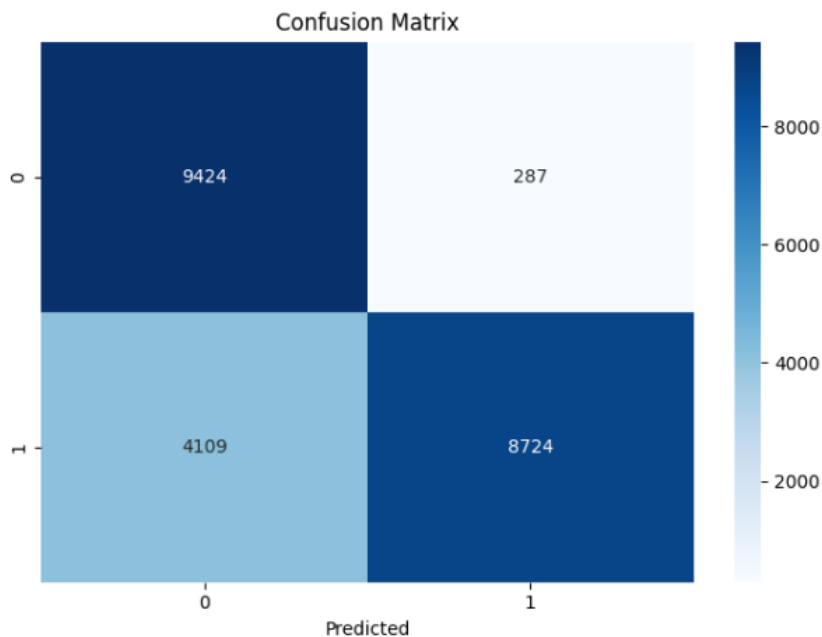


Figure 4.5: Confusion matrix of the CNN-LSTM model on the NSL-KDD test set.

The confusion matrix confirms the behavior observed in the global metrics. The model correctly classified 9,424 normal samples as normal and detected 8,724 attack samples. Only 287 normal samples were incorrectly classified as attacks, which explains the high precision obtained by the system.

However, 4,109 attack samples were classified as normal. These false negatives explain the moderate recall value. From an operational point of view, this result means that the deployed IDS produces reliable alerts, but it may miss some attack attempts. The decision threshold can therefore be adjusted depending on the security policy. A lower threshold may increase attack detection, while a higher threshold may reduce false alarms.

4.3.3 Training Performance Analysis

Figure 4.6 shows the training and validation accuracy and loss curves recorded during the learning phase.

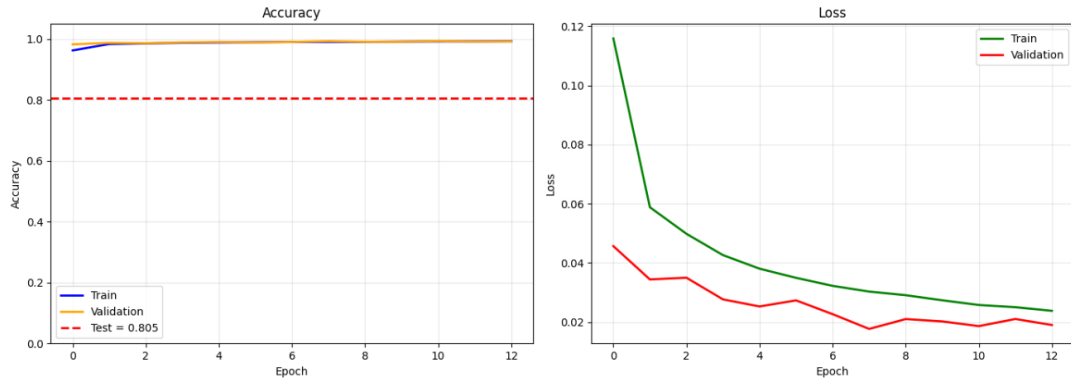


Figure 4.6: Training and validation accuracy and loss curves of the proposed CNN-LSTM model.

The training and validation curves show that the CNN-LSTM model learned rapidly during the first epochs and then stabilized. The validation performance remained close to the training performance, which indicates stable learning behavior during training. The use of early stopping helped avoid unnecessary epochs and allowed the best model weights to be retained. Nevertheless, the independent test performance was lower than the training and validation performance. This difference is expected because the NSL-KDD test set contains unseen and more challenging samples. Therefore, the test results provide a more realistic estimate of the model behavior than the training or validation curves alone.

4.3.4 Comparison of Training, Validation, and Test Metrics

Figure 4.7 compares the training, validation, and test metrics.

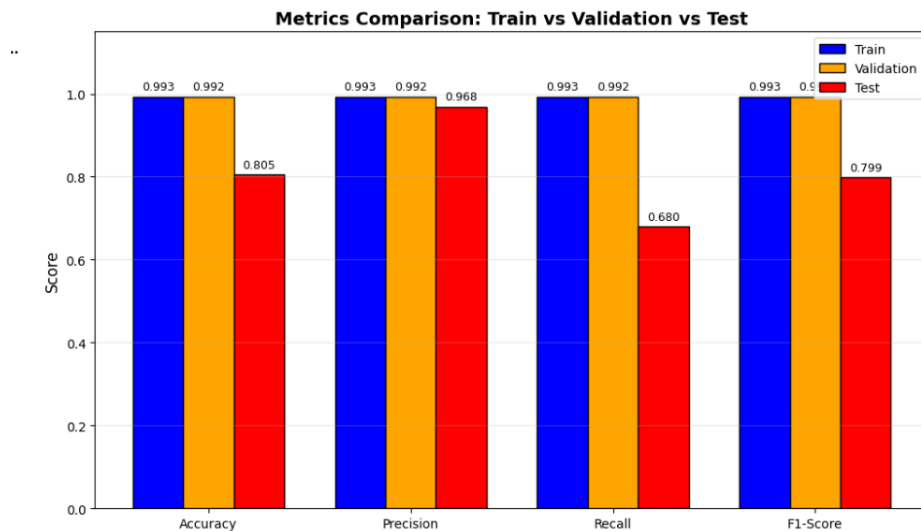


Figure 4.7: Comparison of training, validation, and test performance metrics of the proposed CNN-LSTM model.

The comparison highlights a clear generalization gap between the training/validation sets and the independent test set. While the training and validation metrics are very high, the

final test accuracy is 80.5%. This confirms that the model was evaluated on more difficult traffic patterns during testing. Despite this gap, the model preserved a high precision, which remains an important advantage for real-time alert generation.

4.4 Comparison with Related Work

Table 4.4 compares the proposed CNN–LSTM model with selected deep learning-based IDS studies using the NSL-KDD dataset [26, 32]. The comparison should be interpreted carefully because differences in preprocessing, feature selection, train/test splits, model structure, and decision thresholds can influence the reported metrics.

Table 4.4: Comparison of the proposed CNN–LSTM model with related intrusion detection approaches.

Study	Method	Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Tang et al. (2016)	Deep Learning IDS	NSL-KDD	75.75	98.80	67.80	80.0
Jain et al. (2026)	CNN–LSTM	NSL-KDD	79.36	94.27	69.43	79.0
Our work	CNN–LSTM	NSL-KDD	80.5	96.8	68.0	79.9

The proposed model achieves competitive performance compared with the selected studies. Its main advantage is the high precision value, which indicates a low false alarm tendency. In addition, this work goes beyond offline evaluation by integrating the trained model into a real-time Raspberry Pi deployment. This practical validation is important because it demonstrates the transition from a trained model to an operational embedded IDS prototype.

4.5 Real-Time Detection on Raspberry Pi

4.5.1 Deployment Architecture

After training, the CNN–LSTM model and the preprocessing artifacts were transferred to the Raspberry Pi. The deployment script loads these files before starting packet capture. The main artifacts include the trained model file, the scaler or preprocessing pipeline, the feature column list, the selected feature subset, the categorical encoders, and the label encoder.

The real-time detection chain consists of four main stages. First, Scapy captures packets received from the mirrored switch port. Second, packets are grouped into flows and converted into numerical and categorical statistics similar to the features used in NSL-KDD. Third, the

same preprocessing operations used during training are applied to the live feature vector. Finally, the feature vector is reshaped according to the input format required by the CNN–LSTM model and classified as normal or attack.

```
=====
NSL-KDD Intrusion Detection System -- Raspberry Pi (v6)
=====
Interface   : eth0
BPF filter  : ip
Threshold   : 0.85
Min pkts    : ICMP=1  UDP=2  TCP=2
Promisc mode: ON
Flow expiry : 120.0s idle
Warmup flows: 20 flows (counter blending window)
CSV log     : disabled
=====

[INFO] Loading model: /home/ziad/rasbery/model.h5
[INFO] Model loaded. input_shape=(None, 20, 1)
[INFO] feature_columns : 41 features
[INFO] selected_features: 20 features
[INFO] pipeline steps   : ['feature_columns', 'categorical_columns', 'scaler', 'kbest', 'rfe', 'selected_features']
[INFO] Warming up model ...
[INFO] Ready -- listening for traffic.

TIME                STATUS                CONF        PROTO  SRC                DST
```

Figure 4.8: Initialization of the IDS application and loading of the trained CNN–LSTM model on the Raspberry Pi.

Figure 4.9 presents the real-time processing flow executed on the Raspberry Pi. The improved script is based on flow reconstruction rather than isolated packet classification. This is more consistent with the NSL-KDD representation, since many features describe connection-level behavior instead of individual packets.

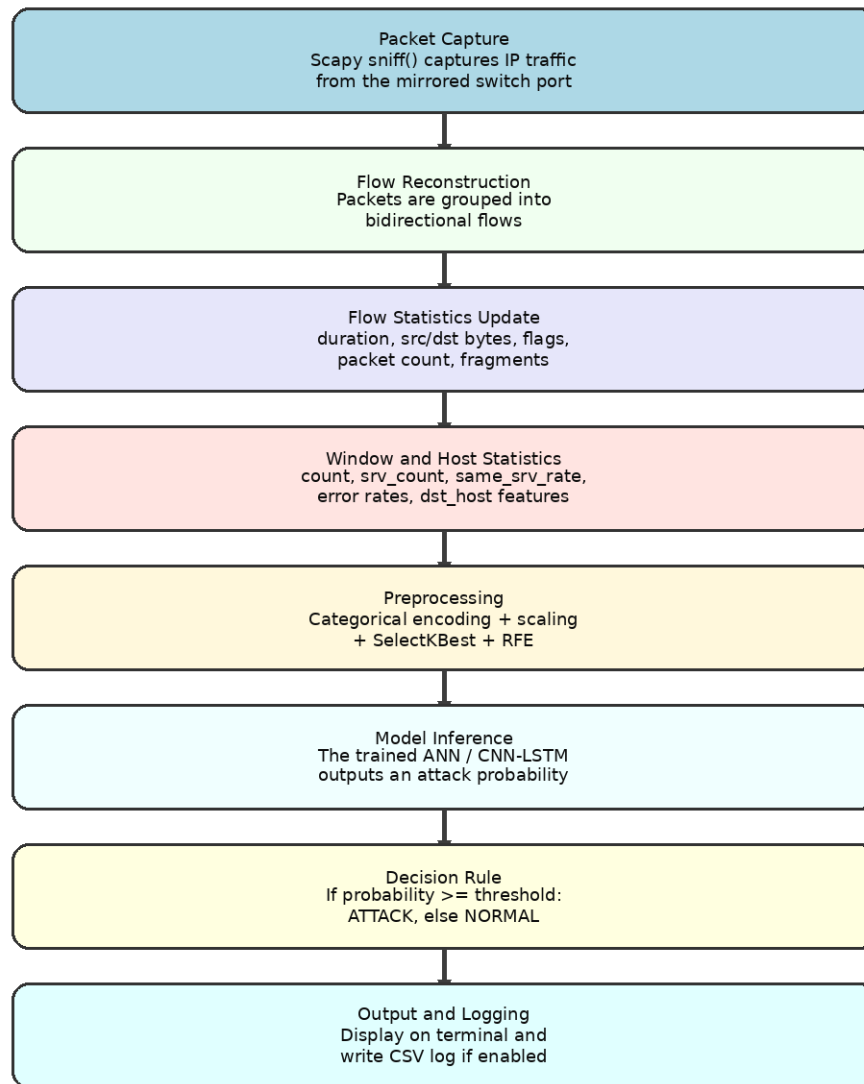


Figure 4.9: Real-time detection flow executed on the Raspberry Pi.

4.5.2 Online Experimental Workflow

The online experiment is summarized in Figure 4.10. Normal traffic was generated by the client laptop, while the Arduino Ethernet board generated repeated HTTP request traffic. All traffic passed through the router and switch. The switch mirrored a copy of the traffic to the Raspberry Pi, where the IDS analyzed the flows in real time and produced classification outputs.

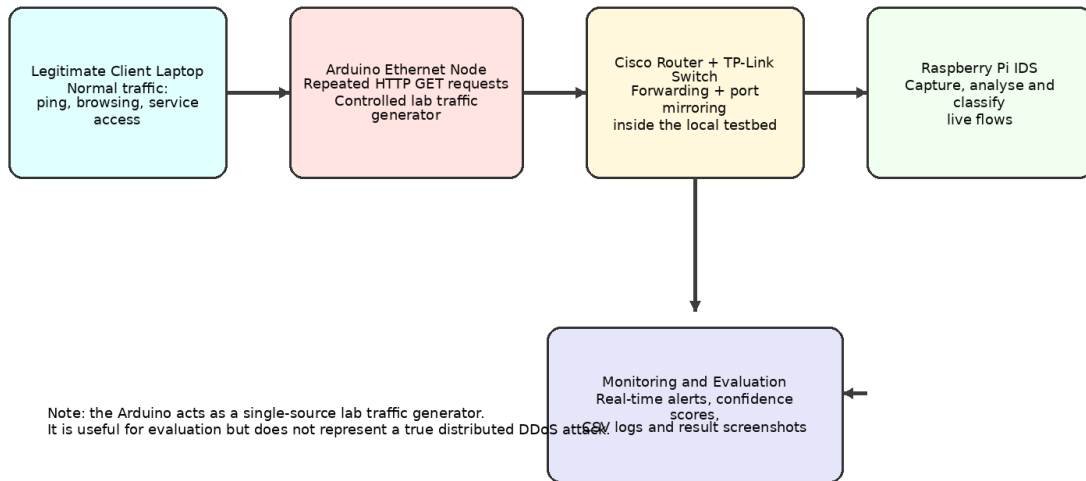


Figure 4.10: Experimental workflow used for live traffic generation, monitoring, and evaluation.

This workflow validates the complete deployment chain implemented in the code: packet capture, flow reconstruction, feature extraction, preprocessing, model inference, alert display, and result logging. It also shows that the Raspberry Pi can execute the trained model in a live monitoring scenario.

4.5.3 Detection of Normal Traffic

Normal ICMP traffic was generated between devices in the local network. The deployed IDS analyzed the captured packets, reconstructed the corresponding flow statistics, and classified the observed traffic as normal. This result confirms that the system can process legitimate traffic without systematically producing alerts.

The decision threshold used during offline experiments was set to 0.30. In the deployment script, this threshold can be configured depending on the desired security policy. When the predicted attack probability remains below the selected threshold, the flow is reported as normal. When it exceeds the threshold, the IDS generates an attack alert.

16:50:29.829	[NORMAL]	conf=76.7%	ICMP	192.168.1.1	-> 192.168.1.15
16:50:30.404	[NORMAL]	conf=76.7%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:30.621	[NORMAL]	conf=78.0%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:31.416	[NORMAL]	conf=77.2%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:31.626	[NORMAL]	conf=77.6%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:32.428	[NORMAL]	conf=77.2%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:32.637	[NORMAL]	conf=77.6%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:33.411	[NORMAL]	conf=77.1%	ICMP	192.168.1.1	-> 192.168.1.15
16:50:33.628	[NORMAL]	conf=76.7%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:33.843	[NORMAL]	conf=78.0%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:34.462	[NORMAL]	conf=76.7%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:34.673	[NORMAL]	conf=78.0%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:35.475	[NORMAL]	conf=77.2%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:35.684	[NORMAL]	conf=77.6%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:36.490	[NORMAL]	conf=77.2%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:36.698	[NORMAL]	conf=77.6%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:37.504	[NORMAL]	conf=77.2%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:37.711	[NORMAL]	conf=77.6%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:38.520	[NORMAL]	conf=77.2%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:38.774	[NORMAL]	conf=77.6%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:39.546	[NORMAL]	conf=77.2%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:39.762	[NORMAL]	conf=77.6%	ICMP	192.168.1.15	-> 192.168.1.13
16:50:40.559	[NORMAL]	conf=77.2%	ICMP	192.168.1.13	-> 192.168.1.15
16:50:40.771	[NORMAL]	conf=77.6%	ICMP	192.168.1.15	-> 192.168.1.13

Figure 4.11: Normal traffic correctly identified by the deployed IDS.

4.5.4 Detection of Abnormal Traffic

To evaluate the response of the IDS to suspicious traffic, the Arduino Ethernet board generated repeated HTTP GET requests toward a local target. The Raspberry Pi captured the mirrored traffic, reconstructed the flow features, and submitted them to the CNN-LSTM model. The deployed IDS generated attack alerts when the prediction confidence exceeded the configured decision threshold.

The alerts displayed during the experiment showed high confidence values for abnormal traffic, with several detections reported in the upper part of the decision range. This confirms that the complete online chain operated correctly: the system captured live traffic, extracted the required features, reused the training preprocessing artifacts, executed model inference, and displayed the resulting decision in real time.

16:55:31.824	[ATTACK DETECTED]	conf=90.3%	TCP	192.168.1.100	-> 192.168.1.1
16:55:32.048	[ATTACK DETECTED]	conf=89.8%	TCP	192.168.1.100	-> 192.168.1.1
16:55:32.273	[ATTACK DETECTED]	conf=89.3%	TCP	192.168.1.100	-> 192.168.1.1
16:55:32.541	[ATTACK DETECTED]	conf=88.6%	TCP	192.168.1.100	-> 192.168.1.1
16:55:32.766	[ATTACK DETECTED]	conf=87.8%	TCP	192.168.1.100	-> 192.168.1.1
16:55:32.988	[ATTACK DETECTED]	conf=86.9%	TCP	192.168.1.100	-> 192.168.1.1
16:55:33.243	[NORMAL]	conf=3.1%	TCP	192.168.1.100	-> 192.168.1.1
16:55:33.489	[ATTACK DETECTED]	conf=85.8%	TCP	192.168.1.100	-> 192.168.1.1
16:55:33.717	[NORMAL]	conf=84.5%	TCP	192.168.1.100	-> 192.168.1.1
16:55:33.950	[NORMAL]	conf=3.0%	TCP	192.168.1.100	-> 192.168.1.1
16:55:34.196	[NORMAL]	conf=3.0%	TCP	192.168.1.100	-> 192.168.1.1
16:55:34.426	[NORMAL]	conf=3.1%	TCP	192.168.1.100	-> 192.168.1.1
16:55:34.650	[NORMAL]	conf=3.1%	TCP	192.168.1.100	-> 192.168.1.1
16:55:34.880	[NORMAL]	conf=3.2%	TCP	192.168.1.100	-> 192.168.1.1
16:55:35.120	[NORMAL]	conf=3.2%	TCP	192.168.1.100	-> 192.168.1.1
16:55:35.348	[NORMAL]	conf=3.3%	TCP	192.168.1.100	-> 192.168.1.1
16:55:35.571	[NORMAL]	conf=3.3%	TCP	192.168.1.100	-> 192.168.1.1
16:55:35.800	[NORMAL]	conf=3.3%	TCP	192.168.1.100	-> 192.168.1.1
16:55:36.052	[NORMAL]	conf=3.4%	TCP	192.168.1.100	-> 192.168.1.1
16:55:36.282	[NORMAL]	conf=3.4%	TCP	192.168.1.100	-> 192.168.1.1
16:55:36.506	[NORMAL]	conf=3.4%	TCP	192.168.1.100	-> 192.168.1.1
16:55:36.736	[NORMAL]	conf=3.4%	TCP	192.168.1.100	-> 192.168.1.1
16:55:36.976	[NORMAL]	conf=1.9%	TCP	192.168.1.1	-> 192.168.1.100
16:55:37.205	[NORMAL]	conf=13.6%	TCP	192.168.1.1	-> 192.168.1.100
16:55:37.429	[NORMAL]	conf=1.9%	TCP	192.168.1.1	-> 192.168.1.100
16:55:37.660	[ATTACK DETECTED]	conf=89.8%	TCP	192.168.1.100	-> 192.168.1.1
16:55:37.902	[ATTACK DETECTED]	conf=89.3%	TCP	192.168.1.100	-> 192.168.1.1
16:55:38.129	[ATTACK DETECTED]	conf=88.6%	TCP	192.168.1.100	-> 192.168.1.1
16:55:38.365	[ATTACK DETECTED]	conf=87.8%	TCP	192.168.1.100	-> 192.168.1.1
16:55:38.600	[ATTACK DETECTED]	conf=86.9%	TCP	192.168.1.100	-> 192.168.1.1
16:55:38.844	[ATTACK DETECTED]	conf=85.8%	TCP	192.168.1.100	-> 192.168.1.1

Figure 4.12: Real-time detection of abnormal network traffic generated during the laboratory experiment.

4.6 Critical Analysis

The obtained results show that the proposed CNN–LSTM IDS provides reliable alert generation, with a precision of 96.8% and only 287 false positives on the NSL-KDD test set. This is an important advantage because intrusion detection systems must avoid overwhelming administrators with excessive false alarms.

The feature selection process also contributed to the practical implementation of the system. Reducing the original NSL-KDD feature space to a selected subset decreased the input dimensionality and simplified the deployment pipeline. This is particularly useful for the Raspberry Pi, where computational resources are more limited than in a cloud training environment.

However, the recall value of 68.0% shows that the model still misses some attacks. This limitation is visible in the number of false negatives observed in the confusion matrix. Future work should therefore investigate threshold adjustment, class balancing, alternative feature selection methods, and more recent datasets to improve attack coverage.

Another important limitation is related to the dataset itself. NSL-KDD is widely used in intrusion detection research, but it does not fully represent modern network traffic or recent attack strategies. For this reason, future validation should include datasets such as UNSW-NB15, CICIDS2017, or CSE-CIC-IDS2018.

The real-time deployment also has practical limits. The Raspberry Pi experiment confirms the feasibility of embedded IDS operation, but additional measurements are needed to characterize performance more completely. Future experiments should include CPU load, memory consumption, packet processing delay, throughput, and longer monitoring periods with more diverse traffic.

4.6.1 Practical Interpretation of the Results

From an operational perspective, the system is well suited for scenarios where alert reliability is important. The high precision means that most generated alerts are meaningful. However, the moderate recall means that the system should not be considered a complete security solution by itself. It should be integrated as one component of a broader monitoring strategy.

The configurable threshold is also important. A lower threshold may increase the number of detected attacks but can produce more false alarms. A higher threshold may reduce false alarms but can miss more attacks. Therefore, the threshold should be selected according to the risk level and administrative capacity of the target environment.

4.7 Conclusion

This chapter presented the implementation, evaluation, and Raspberry Pi deployment of the proposed CNN–LSTM intrusion detection system. The model was trained and evaluated on the NSL-KDD dataset, then exported with its preprocessing artifacts for real-time deployment.

The offline evaluation produced an accuracy of 80.5%, a precision of 96.8%, a recall of 68.0%, and an F1-score of 79.9%. These results show that the model generates reliable attack alerts while maintaining a moderate attack detection rate. The confusion matrix confirmed the low false positive count and also highlighted the presence of false negatives.

The real-time experiment demonstrated that the trained model can be deployed on a Raspberry Pi. The online script captured mirrored traffic, reconstructed flow-based features, applied the saved preprocessing chain, executed CNN–LSTM inference, and generated normal or attack decisions. The Arduino Ethernet board was used as a controlled HTTP traffic generator to evaluate the responsiveness of the system under abnormal traffic conditions.

Overall, the proposed system demonstrates the feasibility of implementing a deep learning-based IDS on a low-cost embedded platform. Although further validation is required using more recent datasets and more diverse live traffic, the implementation confirms that the developed CNN–LSTM model can be integrated into a practical real-time intrusion detection workflow.

General Conclusion and Perspectives

This thesis studied the design, evaluation, and deployment of a deep learning-based anomaly detection system for enterprise networks. The work was motivated by the increasing complexity of modern infrastructures and by the limitations of traditional intrusion detection mechanisms when facing unknown attacks, heterogeneous traffic, and nonlinear relationships between network features.

The first part of the thesis established the technical context. Enterprise networks were presented as heterogeneous infrastructures combining LANs, VLANs, WANs, VPNs, hierarchical campus architectures, and spine-leaf data-center architectures. The analysis showed that anomalies may originate from malicious activities, but also from operational causes such as configuration errors, failures, congestion, or unusual legitimate usage. This observation is important because it explains why intrusion detection cannot rely only on simple traffic thresholds.

The second part introduced the deep learning concepts required for the proposed approach. Neural networks, CNNs, RNNs, LSTMs, activation functions, loss functions, and evaluation metrics were described with emphasis on their role in intrusion detection. The CNN-LSTM architecture was selected because it combines two complementary mechanisms: convolutional layers learn local feature patterns, while LSTM layers model dependencies across the learned representation.

The experimental methodology was based on the NSL-KDD dataset. The data were cleaned, encoded, scaled, and reduced from 41 original features to 20 selected features using a two-stage feature-selection approach. The model was trained with binary cross-entropy and Adam optimization, and its performance was evaluated using accuracy, precision, recall, F1-score, and confusion matrix analysis. This multi-metric evaluation was necessary because accuracy alone is not sufficient for judging an IDS.

The obtained test results showed that the proposed model reached an accuracy of 80.5%, a precision of 96.8%, a recall of 68.0%, and an F1-score of 79.9%. These values indicate that the system generated reliable alerts with a low false-positive tendency. However, the moderate recall shows that a portion of attacks remained undetected. This result highlights a central trade-off in intrusion detection: a model can be configured to produce fewer false alarms, but this may reduce its sensitivity to some attacks. The decision threshold must

therefore be chosen according to the security policy of the target environment.

A major practical contribution of the thesis is the deployment of the trained model on a Raspberry Pi platform. The embedded demonstrator integrated packet capture, preprocessing, feature reconstruction, model inference, and alert generation. This experiment confirmed the feasibility of running a deep learning-based IDS on low-cost hardware in a controlled local network. It also showed that a complete IDS project requires more than an offline classification score; it requires a consistent chain from dataset preparation to operational inference.

Despite these contributions, several limitations remain. The NSL-KDD dataset is useful for benchmarking, but it is old and does not fully represent modern traffic, encrypted flows, cloud services, IoT environments, or recent attack techniques. Moreover, the binary classification used in this work does not distinguish between attack families. The online experiments were also performed in a controlled laboratory environment and should be extended to longer and more diverse traffic captures.

Future work should therefore focus on several directions. First, the model should be evaluated on more recent datasets such as CICIDS2017, CSE-CIC-IDS2018, UNSW-NB15, or other flow-based datasets. Second, multiclass classification should be implemented to identify attack categories instead of only separating normal and malicious traffic. Third, the decision threshold should be optimized according to the desired balance between precision and recall. Fourth, deployment tests should include CPU load, memory usage, packet-processing latency, throughput, and robustness under continuous operation. Finally, explainability methods could be integrated to help administrators understand which traffic features contributed most to each alert.

In conclusion, this thesis demonstrates that a hybrid CNN-LSTM model can provide a practical and effective basis for anomaly detection in enterprise networks. The work also confirms that embedded deployment is feasible when preprocessing artifacts and model components are carefully exported and reused. The proposed system is not a final industrial IDS, but it represents a coherent prototype that connects theoretical modeling, benchmark evaluation, and real-time implementation.

Bibliography

- [1] Karen Scarfone and Peter Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. Special Publication 800-94. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2007. DOI: [10.6028/NIST.SP.800-94](https://doi.org/10.6028/NIST.SP.800-94). URL: <https://csrc.nist.gov/pubs/sp/800/94/final>.
- [2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning.” In: *Nature* 521.7553 (2015), pp. 436–444. DOI: [10.1038/nature14539](https://doi.org/10.1038/nature14539).
- [3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. URL: <https://www.deeplearningbook.org/>.
- [4] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory.” In: *Neural Computation* 9 (1997), pp. 1735–1780. DOI: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735).
- [5] Mahbod Tavallaee et al. “A Detailed Analysis of the KDD CUP 99 Dataset.” In: *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications*. 2009. DOI: [10.1109/CISDA.2009.5356528](https://doi.org/10.1109/CISDA.2009.5356528).
- [6] Canadian Institute for Cybersecurity. *NSL-KDD Dataset*. Online. Accessed: 2026-06-16. 2009. URL: <https://www.unb.ca/cic/datasets/nsll.html>.
- [7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: A Survey.” In: *ACM Computing Surveys* 41.3 (July 2009), 15:1–15:58.
- [8] A. Aldweesh, A. Derhab, and A. Z. Emam. “Deep Learning Approaches for Anomaly-Based Intrusion Detection Systems: A Survey, Taxonomy, and Open Issues.” In: *Knowledge-Based Systems* 189 (2020), p. 105124. DOI: [10.1016/j.knosys.2019.105124](https://doi.org/10.1016/j.knosys.2019.105124).
- [9] Zeeshan Ahmad et al. “Network Intrusion Detection System: A Systematic Study of Machine Learning and Deep Learning Approaches.” In: *Transactions on Emerging Telecommunications Technologies* 32.1 (2021), e4150.
- [10] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. 7th ed. Hoboken, NJ, USA: Pearson, 2017.
- [11] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th ed. Upper Saddle River, NJ, USA: Pearson, 2011.

- [12] Cisco Systems. *Campus Network for High Availability Design Guide*. Tech. rep. San Jose, CA, USA: Cisco Systems, 2008.
- [13] Cisco Systems. *Cisco SD-WAN Architecture Overview*. White Paper. San Jose, CA, USA: Cisco Systems, 2020.
- [14] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. “A Scalable, Commodity Data Center Network Architecture.” In: *Proceedings of the ACM SIGCOMM Conference*. 2008, pp. 63–74.
- [15] William Stallings. *Network Security Essentials: Applications and Standards*. 6th ed. Hoboken, NJ, USA: Pearson, 2017.
- [16] Scott Rose et al. *Zero Trust Architecture*. Special Publication 800-207. Gaithersburg, MD, USA: National Institute of Standards and Technology, Aug. 2020.
- [17] IEEE. *IEEE Standard for Ethernet*. IEEE, 2018.
- [18] IEEE. *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks*. IEEE, 2018.
- [19] Markus Ring et al. “A Survey of Network-Based Intrusion Detection Data Sets.” In: *Computers & Security* 86 (2019), pp. 147–167.
- [20] Ravi Vinayakumar et al. “Deep Learning Approach for Intelligent Intrusion Detection System.” In: *IEEE Access* 7 (2019), pp. 41525–41550.
- [21] Canadian Institute for Cybersecurity. *Intrusion Detection Evaluation Dataset: CICIDS2017*. Online. Accessed: 2026-06-16. 2017. URL: <https://www.unb.ca/cic/datasets/ids-2017.html>.
- [22] Canadian Institute for Cybersecurity. *CSE-CIC-IDS2018 on AWS*. Online. Accessed: 2026-06-16. 2018. URL: <https://www.unb.ca/cic/datasets/ids-2018.html>.
- [23] UNSW Canberra. *The UNSW-NB15 Dataset*. Online. Accessed: 2026-06-16. 2021. URL: <https://research.unsw.edu.au/projects/unsw-nb15-dataset>.
- [24] M. Abdallah et al. “A Hybrid CNN-LSTM Based Approach for Anomaly Detection Systems in SDNs.” In: *Proceedings of ARES 2021*. 2021.
- [25] S. S. Bamber et al. “A Hybrid CNN-LSTM Approach for Intelligent Cyber Intrusion Detection System.” In: *Computers & Security* 148 (2025), p. 104146. DOI: [10.1016/j.cose.2024.104146](https://doi.org/10.1016/j.cose.2024.104146).
- [26] V. Jain et al. “Optimizing Intrusion Detection Using a Hybrid CNN-LSTM Deep Learning Framework on the NSL-KDD Dataset.” In: *Discover Computing* 29.1 (2026), p. 194. DOI: [10.1007/s10791-026-10056-6](https://doi.org/10.1007/s10791-026-10056-6).

- [27] TensorFlow. *tf.keras.losses.BinaryCrossentropy*. Online documentation. Accessed: 2026-06-16. 2024. URL: https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy.
- [28] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *International Conference on Learning Representations*. 2015. arXiv: [1412.6980](https://arxiv.org/abs/1412.6980) [cs.LG].
- [29] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python.” In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [30] scikit-learn developers. *Feature Selection*. Online documentation. Accessed: 2026-06-16. 2026. URL: https://scikit-learn.org/stable/modules/feature_selection.html.
- [31] Keras. *Optimizers*. Online documentation. Accessed: 2026-06-16. 2026. URL: <https://keras.io/api/optimizers/>.
- [32] T. A. Tang et al. “Deep Learning Approach for Network Intrusion Detection in Software Defined Networking.” In: *Proceedings of the 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. Fez, Morocco, 2016, pp. 258–263. DOI: [10.1109/WINCOM.2016.7777224](https://doi.org/10.1109/WINCOM.2016.7777224).