From industrial process control to network security: deployable data-driven approaches for anomaly and fault detection

Italian National Ph.D. Program in Autonomous Systems

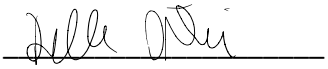
ACADEMIC DISCIPLINE: SYSTEMS AND CONTROL ENGINEERING (IINF-04/A)

Final Dissertation

From Industrial Process Control to Network Security: Deployable Data-driven approaches for Anomaly and Fault Detection

by

Daniele Antonucci



Administrative Headquarters:

Politecnico di Bari – Department of Electrical and Information Engineering

Hosting University:

University of Parma – Department of Engineering and Architecture

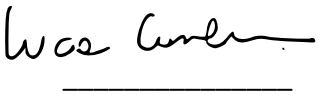
Referees:

Prof. Enrico Magli

Prof. Cristiano Maria Verrelli

Supervisors:

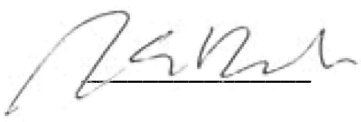
Prof. Luca Consolini



Prof. Gianluigi Ferrari

Coordinator of Ph.D Program

Prof. Mariagrazia Dotoli



Course no. 38, 01/11/2022-31/10/2025

LIBERATORIA PER L'ARCHIVIAZIONE DELLA TESI DI DOTTORATO

Al Magnifico Rettore
del Politecnico di Bari

Il sottoscritto Antonucci Daniele nato a Caserta il 25/01/1997
residente a Fiorenzuola d'arda in via Manfredi 26 e-mail danieleantonucci97@gmail.com
iscritto al 3° anno di Corso di Dottorato di Ricerca in Autonomous Systems ciclo 38
ed essendo stato ammesso a sostenere l'esame finale con la prevista discussione della tesi dal titolo:
From Industrial Process Control to Network Security: Deployable Data-driven approaches for Anomaly and Fault Detection

DICHIARA

1) di essere consapevole che, ai sensi del D.P.R. n. 445 del 28.12.2000, le dichiarazioni mendaci, la falsità negli atti e l'uso di atti falsi sono puniti ai sensi del codice penale e delle Leggi speciali in materia, e che nel caso ricorressero dette ipotesi, decade fin dall'inizio e senza necessità di nessuna formalità dai benefici conseguenti al provvedimento emanato sulla base di tali dichiarazioni;
2) di essere iscritto al Corso di Dottorato di ricerca Autonomous Systems ciclo 38 corso attivato ai sensi del "*Regolamento dei Corsi di Dottorato di ricerca del Politecnico di Bari"*, emanato con D.R. n.286 del 01.07.2013;
3) di essere pienamente a conoscenza delle disposizioni contenute nel predetto Regolamento in merito alla procedura di deposito, pubblicazione e autoarchiviazione della tesi di dottorato nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica;
4) di essere consapevole che attraverso l'autoarchiviazione delle tesi nell'Archivio Istituzionale ad accesso aperto alla letteratura scientifica del Politecnico di Bari (IRIS-POLIBA), l'Ateneo archivierà e renderà consultabile in rete (nel rispetto della Policy di Ateneo di cui al D.R. 642 del 13.11.2015) il testo completo della tesi di dottorato, fatta salva la possibilità di sottoscrizione di apposite licenze per le relative condizioni di utilizzo (di cui al sito http://www.creativecommons.it/Licenze), e fatte salve, altresì, le eventuali esigenze di "embargo", legate a strette considerazioni sulla tutelabilità e sfruttamento industriale/commerciale dei contenuti della tesi, da rappresentarsi mediante compilazione e sottoscrizione del modulo in calce (Richiesta di embargo);
5) che la tesi da depositare in IRIS-POLIBA, in formato digitale (PDF/A) sarà del tutto identica a quelle **consegnate**/inviate/da inviarsi ai componenti della commissione per l'esame finale e a qualsiasi altra copia depositata presso gli Uffici del Politecnico di Bari in forma cartacea o digitale, ovvero a quella da discutere in sede di esame finale, a quella da depositare, a cura dell'Ateneo, presso le Biblioteche Nazionali Centrali di Roma e Firenze e presso tutti gli Uffici competenti per legge al momento del deposito stesso, e che di conseguenza va esclusa qualsiasi responsabilità del Politecnico di Bari per quanto riguarda eventuali errori, imprecisioni o omissioni nei contenuti della tesi;
6) che il contenuto e l'organizzazione della tesi è opera originale realizzata dal sottoscritto e non compromette in alcun modo i diritti di terzi, ivi compresi quelli relativi alla sicurezza dei dati personali; che pertanto il Politecnico di Bari ed i suoi funzionari sono in ogni caso esenti da responsabilità di qualsivoglia natura: civile, amministrativa e penale e saranno dal sottoscritto tenuti indenni da qualsiasi richiesta o rivendicazione da parte di terzi;
7) che il contenuto della tesi non infrange in alcun modo il diritto d'Autore né gli obblighi connessi alla salvaguardia di diritti morali od economici di altri autori o di altri aventi diritto, sia per testi, immagini, foto, tabelle, o altre parti di cui la tesi è composta.

Luogo e data PARMA, 16/01/26 Firma Daniele Antonucci

Il/La sottoscritto, con l'autoarchiviazione della propria tesi di dottorato nell'Archivio Istituzionale ad accesso aperto del Politecnico di Bari (POLIBA-IRIS), pur mantenendo su di essa tutti i diritti d'autore, morali ed economici, ai sensi della normativa vigente (Legge 633/1941 e ss.mm.ii.),

CONCEDE

- al Politecnico di Bari il permesso di trasferire l'opera su qualsiasi supporto e di convertirla in qualsiasi formato al fine di una corretta conservazione nel tempo. Il Politecnico di Bari garantisce che non verrà effettuata alcuna modifica al contenuto e alla struttura dell'opera.
- al Politecnico di Bari la possibilità di riprodurre l'opera in più di una copia per fini di sicurezza, back-up e conservazione.

Luogo e data PARMA, 16/01/26 Firma Daniele Antonucci

Daniele Antonucci

From Industrial Process Control to Network Security: Deployable Data-driven approaches for Anomaly and Fault Detection

Thesis submitted for the degree of Philosophiae Doctor

Italian National Ph.D. Program in Autonomous Systems
University of Parma

Tutors
Prof. Engr. *Luca Consolini*
Prof. Engr. *Gianluigi Ferrari*

UNIVERSITÀ
DI PARMA

2025

The doctoral scholarship was funded by the European Union - Next Generation EU, Mission 4 Component 2 Investment 3.3 CUP D93D22001390001 - Decree No. 352 (09th April 2022) of Italian Ministry of University and Research - Concession Decree No. 2153 (28th December 2022) of the Italian Ministry of University and Research, within the Italian National Program PhD Programme in Autonomous Systems (DAuSy).

Thesis submitted for the degree of *Philosophiae Doctor*
Italian National Ph.D. Program in Autonomous Systems

Cycle:
38th

Administrative Headquarters:
Politecnico di Bari

Hosting University:
University of Parma

Title:
From Industrial Process Control to Network Security: Deployable Data-driven approaches for Anomaly and Fault Detection

Ph.D Candidate:
Daniele Antonucci, University of Parma (Parma, Italy)

Tutors:
Prof. Engr. Luca Consolini, University of Parma (Parma, Italy)
Prof. Engr. Gianluigi Ferrari, University of Parma (Parma, Italy)

Coordinator:
Prof. Engr. Mariagrazia Dotoli, Politecnico di Bari (Bari, Italy)

External Reviewers:
Prof. Enrico Magli, Polytechnic of Turin (Turin, Italy)
Prof. Cristiano Maria Verrelli, University of Rome Tor Vergata (Rome, Italy)

Last version:
January 12th, 2026

We are continually faced with great opportunities brilliantly disguised as insoluble problems.
Lee Iacocca

To everyone who made this journey possible.

Contents

List of Figures

List of Tables

Acronyms

1D one-dimensional. 7, 34, 35, 52

AE Auto Encoder. vi, 3, 9, 21, 22

AI Artificial Intelligence. xiv, 1–4, 8, 10, 46, 47, 52, 54, 59, 63, 70, 88

ALSTM Aggregated LSTM. 34

ANN Artificial Neural Network. 5, 34

API Application Programming Interface. 71, 72

AQI Air Quality Index. 33, 35

ARIMA AutoRegressive Integrated Moving Average. 3

AUC Area Under the Curve. 28, 62, 64

AUCROC Area Under the Curve Receiver Operating Characteristic. viii, 28–30, 62

BiGRU Bidirection GRU. vii, 33–35, 40–42, 44–51, 70, 72, 73, 82–88

BiLSTM Bidirection LSTM. 33–35, 40, 41, 45, 47, 49, 50, 70, 72–74, 82–87

BiRNN Bidirectional RNN. 37

BO Bayesan Optimization. 25, 26, 29, 37–39, 61, 69

CBGRU Convolutional and BiGRU. 34

CBLOF Clustering Based Local Outlier Factor. 29, 30

CIP Clean In Place. 12

CNN Convolutional Neural Network. vii, 6

CNNA Convolutional Neural Network with Attention. 73

COF Connectivity-based Outlier Factor. 29, 30

COPOD COPula based Outlier Detection. 29, 30

CPP Critical Process Parameter. 12

CQA Critical Quality Attributes. 12

DAGMM Deep Autoencoding Gaussian Mixture Model. 29, 30

DBSCAN Density-Based Spatial Clustering of Applications with Noise. vi, 17, 18

DDoS Distributed Denial of Service. 52, 54, 58, 71, 73–76

DL Deep Learning. vi–ix, xiv, 1, 3, 4, 7–10, 21, 22, 26, 30, 33–35, 37–40, 42, 43, 45–50, 52–54, 58, 60, 61, 63–67, 69, 70, 72, 73, 76–80, 82, 85–89

DNN Deep Neural Network. 22

DoS Denial of Service. 52, 56–58, 74, 75

DT Decision Tree. 35, 37, 54, 73, 74, 77

ECOD Empirical-Cumulative-distribution-based Outlier Detection. 29, 30

EI Expected Improvement. 27, 39

FL Federated Learning. 53

FN False Negative. 61, 62, 80, 81

FP False Positive. 61, 62, 81

FP32 32-bit Floating Point. vii–ix, 63, 64, 66–69, 79, 80, 82, 83, 85–87

FPR False Positive Rate. 28

FTIR Fourier Transform InfraRed. 13

GA Genetic Algorithm. 73

GBTR Gradient Boosted Tree Regression. 34

GP Gaussian Process. viii, 26, 27, 38–40

GRU Gated Recurrent Unit. 6, 9, 33–35, 40, 41, 45, 47, 49, 50, 52, 63, 64, 69, 70, 72–74, 82–87

GSK GlaxoSmithKlaine. viii, 12, 21, 23, 30

HBOS Histrogram-based Outlier Detection. 29, 30

HI Health Indicator. vi, 9, 11, 12

HPO HyperParameter Optimization. 25, 29, 35, 39, 60, 61, 74, 78

IdM Identity Management. 52

IDS Intrusion Detection System. vii, 52

IF Isolation Forest. vi, 3, 18, 29, 30

IG Information Gain. 73

INT8 8-bit INTeger. vii–ix, 63, 64, 66–70, 79, 82–87

IoT Internet of Things. vii, xiv, 1–4, 8, 9, 33, 35, 42, 46, 47, 52–58, 64, 69–71, 88

k-NN k-Nearest Neighbors. 3, 29, 30, 34

LGBM Light Gradient Boosting Machine. 53, 54

LMU Legendre Memory Unit. 33, 36, 37, 41, 42, 47, 50

LOF Local Outlier Factor. 29, 30

LoRaWAN Long Range Wide Area Network. 52, 54

LSTM Long Short-Term Memory. xiv, 3, 5, 6, 9, 21–23, 25, 26, 29, 30, 33–35, 40, 41, 45, 47, 49, 50, 52–54, 63, 64, 69, 70, 72–74, 82–88

MACC Multiply and ACCumulate. viii, 42, 45, 47, 50, 63, 65

MAD Mean Absolute Deviation. 21, 22, 27, 28

MAE Mean Absolute Error. vii, 33, 34, 41, 42, 46

MAPE Mean Absolute Percentage Error. vii, 33, 41, 42, 46

MCC Matthews Correlation Coefficient. 62, 64

MD Mahalanobis Distance. 27, 30

mGRU modified GRU. 53, 54

MitM Man-in-the-Middle. 57

ML Machine Learning. vi–viii, xiv, 1, 3, 4, 9, 12, 20, 21, 25, 26, 30, 33–35, 37, 39, 40, 42, 43, 45, 47–50, 52, 53, 60, 63, 73, 77

MLP Multi-Layer Perceptron. 34, 52, 63, 64, 66–68

MSE Mean Squared Error. viii, 26–28, 30

NB NarrowBand. 52

NCAS Normalized Composite Accuracy-Size. vii, 10, 81, 86–88

NIR Near InfraRed. 13

NN Neural Network. 5

NOC Normal Operating Condition. 23

NOS Network Operating System. 71

ODE Ordinary Differential Equation. 36

OVS Open vSwitch. viii, 74–76

PC Principal Component. 14

PCA Principal Component Analysis. xiv, 4, 9, 13–15, 18, 21–23, 29, 30, 34, 35, 88

PCC Pearson Correlation Coefficient. vi, 37, 38

PLS Partial Least Squares. 13, 21

PM Particulate Matter. vii, 33

PRIQ PRuning and Integer Quantization. vii, 74, 80, 82, 87

PTQ Post-Training Quantization. 8, 9, 33, 47, 48, 51–53, 64, 65, 69, 80, 87, 88

PTQA Post-Training Quantization Aware. vii, 74, 79

QbD Quality by Design. 12, 13

R2L Remote-to-Local. 76

R^2 coefficient of determination. 33, 41, 42

RAM Random Access Memory. 47, 51, 63, 65–69, 85

RF Random Forest. viii, 34, 35, 37, 39, 40, 53, 54, 73, 77

RFE Recursive Feature Elimination. 73

RMSE Root Mean Squared Error. vi, vii, 33, 34, 39, 41–43, 46

RNN Recurrent Neural Network. 3

ROC Receiver Operating Characteristic. 62, 64

SARIMAX Seasonal AutoRegressive Integrated Moving Average with eXogenous regressor. 34

SavGol Savitzky-Golay. vi, 19

SDN Software-Defined Networking. 9, 52, 70–78, 87, 88

SGD Stochastic Gradient Descent. 54

SMBO Sequential Model-Based Optimization. 26

SMOTE Synthetic Minority Oversampling Technique. 53, 77

SOD Subspace Outlier Detection. 29, 30

SPE Squared Prediction Error. 22

SVDD Support Vector Data Description. 22, 29, 30

SVM Support Vector Machine. 3, 22, 29, 30, 34, 54

SVR Support Vector Regression. 34

TCN Temporal Convolutional Network. 7, 9, 33, 37, 41, 42, 47, 50, 53, 63, 64, 66–69, 88

TF TensorFlow. viii, ix, 82, 83, 85

TFLite TensorFlow Lite. vii–ix, 47–50, 64, 65, 82–84

TN True Negative. 61, 62, 76, 81

TP True Positive. 58, 61, 62, 80, 81

TPR True Positive Rate. 28

VAE Variational Autoencoder. 3

VM Virtual Machine. 74

WCSS Within-Cluster Sum of Squares. 16

WHO World Health Organization. 33

WP Weight Pruning. 8, 10, 70, 74, 79, 80, 87

XSS Cross-Site Scripting. 57, 76

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* in Autonomous Systems and presents the main scientific results achieved during my doctoral studies. The work presented herein is not merely a sequence of studies, but reflects a thematic and methodological evolution motivated by the goal of addressing pressing challenges in the fields of Industry 4.0 and the IoT.

The motivation behind this research stemmed from my curiosity about how data-driven methods could transform industrial systems and redefine the way we approach data analysis and anomaly detection. The first part of this thesis embodies this perspective, focusing on anomaly detection in critical pharmaceutical processes such as lyophilization and batch production to make industrial systems more efficient, reliable, and autonomous. The objective was to develop a model capable not only of identifying faults but also of anticipating them, thereby transforming raw data into actionable indicators of a system's health. Starting from traditional data-driven approaches, such as Principal Component Analysis (PCA) [1], and extending the study to mathematical models based on explicit knowledge of the governing physical laws—often incomplete or built upon simplifying assumptions—the advantages of data-driven methods were highlighted. Unlike traditional methods, data-driven models can learn relationships directly from the data itself, without requiring prior knowledge of the underlying physical mechanisms. This flexibility makes them particularly effective in handling real-world industrial systems, where uncertainties, noise, and unmodeled dynamics are common and difficult to capture through purely analytical formulations.

The next stage of my research focused on enhancing data-driven methods through the use of Deep Learning (DL) models, capable of both detecting and predicting anomalies. Unlike linear data-driven approaches, DL models can learn complex, non-linear relationships within industrial processes, providing a more comprehensive understanding of system behavior. This led to the development of an LSTM-based framework [2] specifically designed to capture temporal dependencies in multivariate signals, enabling accurate time-series forecasting and effective distinction between normal and anomalous operating conditions.

However, building accurate models represents only one side of the challenge. The central question driving the second part of this thesis became: how can these models be effectively deployed on low-cost, resource-limited devices? This perspective shifted the focus from application-specific performance to deployment feasibility. Consequently, the following studies explore the critical balance between model accuracy and computational efficiency. Several ML and DL architectures were evaluated and compared [3], with particular attention to the impact of compression techniques such as quantization and pruning. The research emphasis thus evolved from understanding what a model can achieve to determining how it can do so efficiently in real-world, constrained environments.

Each chapter, based on a distinct scientific study, represents a distinct step in this research trajectory. Taken together, this work reflects not only a contribution to the scientific field but also my personal journey toward understanding how AI can become both powerful and sustainable—an enabler for the next generation of intelligent and autonomous systems.

The full list of papers written by the author is reported hereafter.

List of Papers Written by the Author

[1] Antonucci, D., Bonanni, D., Consolini, L., and Ferrari, G., "Operational optimisation of pharmaceutical lyophilizers through leak detection data analysis," in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, 2024, pp. 51–56. DOI: 10.1109/CASE59546.2024.10711412.

[2] Antonucci, D., Bonanni, D., Palumberi, D., Consolini, L., and Ferrari, G., "Enhancing pharmaceutical batch processes monitoring with predictive lstm-based framework," in *Proceeding of the 22nd International Conference on Informatics in Control, Automation and Robotics*, vol. 1, 2025, pp. 15–24.

[3] Mazinani, A., Antonucci, D., Pietro Pau, D., Davoli, L., and Ferrari, G., "Air quality prediction via embedded ml/dl and quantized models," *IEEE Access*, vol. 13, pp. 154 203–154 218, 2025. DOI: 10.1109/ACCESS.2025.3603920.

Chapter 1

Introduction

1.1 Motivation and Background

The rapid evolution of digital technologies has profoundly reshaped the industrial landscape, introducing new paradigms in automation, data management, and decision-making. Traditional manufacturing systems, once characterized by rigid workflows and limited adaptability, are progressively evolving into intelligent ecosystems capable of self-monitoring, optimization, and predictive control [1]. This transformation is driven by the integration of cyber-physical systems, the Internet of Things (IoT), and Artificial Intelligence (AI), enabling seamless interaction between physical and digital domains.

The motivation behind this research stems from the growing need to ensure **reliability, efficiency, and autonomy** in modern industrial environments. As production systems become increasingly data-centric, vast amounts of heterogeneous and high-dimensional data are continuously generated by machines, sensors, and control systems. Extracting actionable insights from these data streams is crucial for detecting anomalies, preventing failures, and optimizing performance. However, traditional model-based approaches, which rely on explicit physical assumptions or simplified system representations, often fail to capture the complexity, non-linearity, and uncertainty typical of real-world industrial processes. Consequently, data-driven and learning-based methods have emerged as powerful alternatives [2], capable of learning complex patterns directly from data without requiring prior physical knowledge. Yet, their practical deployment within industrial contexts remains challenging—particularly in embedded and IoT-based environments where computational and memory resources are limited. This dual challenge—achieving high predictive performance while maintaining computational feasibility—constitutes the core motivation and guiding principle of the present research [3]. This thesis focuses on the development of efficient and adaptive AI models designed to operate reliably under constrained conditions.

1.2 Industry 4.0

Industry 4.0 represents the fusion of digital technologies with manufacturing systems (as shown in Figure 1.1), enabling real-time communication and decision-making through IoT-based interconnectivity. Machines, sensors, and control units exchange data seamlessly, supporting intelligent automation and self-optimization across production lines. Technologies such as AI, Big Data analytics, Cloud Computing, and Digital Twins collectively enable the transformation from traditional factories to smart and adaptive environments [4]. The ultimate goal is to achieve flexibility, efficiency, and sustainability while maintaining high-quality standards throughout the value chain.

Starting from raw sensor data, Industry 4.0 leverages advanced analytics to convert continuous data streams into meaningful knowledge [5]. Real-time monitoring and integrated data management allow the identification of inefficiencies and the anticipation of maintenance needs. This data-driven transformation sets the foundation for IoT and edge intelligence, where computational capabilities move closer to the source of data—bridging the gap between centralized AI and distributed industrial systems. In sectors ranging from advanced manufacturing to environmental monitoring and critical network infrastructures, modern systems generate vast quantities of data at an unprecedented rate [6]. Data-driven approaches, particularly those based on Machine Learning (ML) and Deep Learning (DL), have emerged as powerful tools for transforming this raw data into actionable insights, enabling deep analysis and predictive approaches. Deploying these data-driven models is often challenged by the complex, high-dimensional nature of the system and the severe resource constraints of edge and embedded devices. Figure 1.2 illustrates a typical data flow, starting from raw machine data and progressing toward model outputs and

Figure 1.1: Conceptual representation of Industry 4.0, illustrating the integration of cyber-physical systems, Internet of Things (IoT), and smart manufacturing technologies within a connected industrial environment.

corresponding actions. It also highlights the potential for model quantization, enabling the deployment of lightweight models on edge devices.

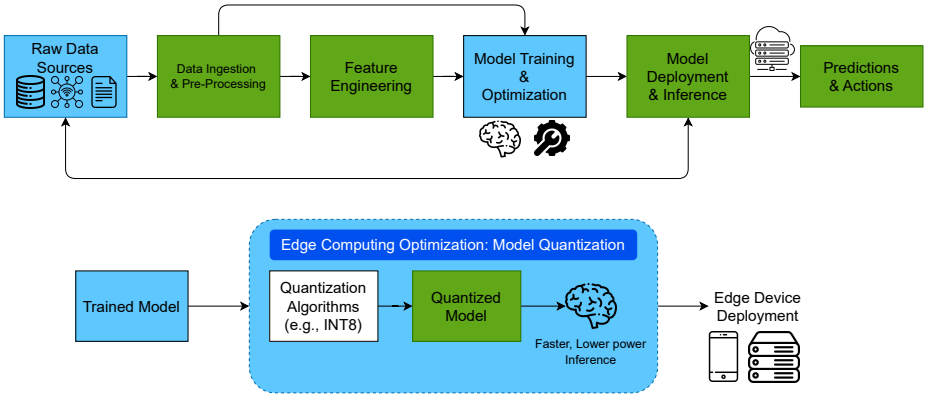


Figure 1.2: A general data flow for a data-driven model, including an optimization step for faster, lower-power inference on edge computing devices.

A sustainable approach within Industry 4.0 emphasizes the integration of smart technologies to minimize environmental impact while maintaining high production efficiency. By combining tools such as AI, IoT, and Digital Twins, industries can monitor energy use, reduce waste, and optimize resource allocation throughout the production chain. Moreover, as highlighted in [7], the convergence of these technologies has the potential to address global challenges like climate change, pollution, and resource scarcity by promoting cleaner, data-driven operations. Through automation, real-time monitoring, and predictive analytics, Industry 4.0 enables companies to transition from reactive to proactive sustainability strategies—improving both environmental performance and economic viability, while paving the way toward Industry 5.0, where human collaboration and ecological responsibility become central priorities.

1.2.1 Pharmaceutical industry

In highly regulated sectors such as the pharmaceutical industry, even small deviations in a manufacturing process can put a risk to the quality of an entire batch, resulting in severe financial

losses and potential safety concerns [8]. Likewise, in the context of our increasingly interconnected world, ensuring cybersecurity relies on the ability to detect malicious behaviors hidden within complex network traffic patterns in real time.

Recent advances in DL have shown remarkable potential for capturing the intricate, non-linear, and temporal dynamics that characterize these systems. State-of-the-art anomaly detection models—such as Auto Encoder (AE), Variational Autoencoder (VAE), Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks [9]—have achieved outstanding accuracy in identifying subtle deviations from normal operational behavior across industrial and cybersecurity domains.

1.2.2 Anomaly Detection in Industrial Systems

Common anomaly detection models span a variety of statistical, machine learning, and DL approaches, each suited to different types of data and applications. Traditional statistical methods, such as Z-score, moving average, or AutoRegressive Integrated Moving Average (ARIMA)-based techniques, identify anomalies by assuming a known data distribution and flagging values that deviate significantly from expected patterns [10]. ML models, including k-Nearest Neighbors (k-NN), Isolation Forest (IF), and One-Class Support Vector Machine (SVM), learn the normal behavior of data and detect deviations without strong assumptions about underlying distributions. More recently, DL approaches like Auto Encoder (AE), Long Short-Term Memory (LSTM) networks, and Variational Autoencoder (VAE) have gained popularity, particularly for time-series and high-dimensional data, as they can capture complex non-linear relationships and temporal dependencies [11]. These models are widely applied across domains such as industrial monitoring, cybersecurity, healthcare, and finance, offering a flexible and powerful toolkit for detecting rare or unexpected events.

1.3 Internet of Things (IoT) and Edge Computing

The Internet of Things (IoT) constitutes the backbone of Industry 4.0, interconnecting a vast number of devices, machines, and systems that continuously generate and exchange data. These devices—often characterized by constrained computational power, limited energy availability, and intermittent connectivity—require efficient mechanisms for local processing and communication. Traditional cloud-centric architectures, although powerful in terms of scalability, are often limited by latency, bandwidth, and privacy concerns. Applications in different scenarios include:

- Smart homes: Devices such as thermostats, lights, and security cameras adjust automatically based on user habits or environmental conditions.
- Healthcare: Wearable devices monitor vital signs and alert doctors or patients to anomalies.
- Industrial automation: Sensors track machinery performance, predict maintenance needs, and optimize production lines.
- Agriculture: Soil moisture sensors and weather monitors help optimize irrigation and crop management.
- Transportation: Connected vehicles provide traffic updates, optimize routes, and enhance safety features.

Edge computing has emerged as a complementary paradigm, where data processing and analysis occur closer to the source of generation, directly on embedded or edge nodes [12]. This approach reduces communication overhead, enhances responsiveness, and increases system robustness. In industrial environments, edge intelligence enables real-time monitoring and control, even in conditions of unreliable network connectivity. Nonetheless, the limited resources of such devices introduce new research challenges related to model efficiency, compression, and energy-aware design.

The integration of AI into tiny IoT devices represents a significant advancement, enabling these devices to process data locally, make intelligent decisions, and operate autonomously. A

major challenge in this context is ensuring that AI models perform effectively while respecting the strict resource constraints of such devices, including limited processing power, memory, and execution time [13]. To address these challenges, evaluating AI models requires considering not only predictive accuracy but also computational efficiency. Achieving a balance between accuracy and resource usage is essential to ensure reliable and effective operation in constrained environments. Techniques such as model quantization, hyperparameter optimization, and the adoption of lightweight architectures are commonly employed to reduce model complexity without compromising performance. These strategies are critical for deploying AI in energy-efficient IoT systems [14].

This thesis builds upon these challenges by exploring learning architectures that can be **efficiently deployed on IoT and edge platforms**, maintaining a balance between accuracy, resource consumption, and real-time performance.

1.4 Data-driven models

Data-driven models have become essential components in modern analysis frameworks, enabling systems to automatically discover patterns, relationships, and structures from complex datasets. In industrial and IoT applications, these models are employed to uncover complex relationships among variables, identify operational patterns, and support predictive and adaptive decision-making. By leveraging historical and real-time data, data-driven approaches allow systems to continuously learn and improve performance under dynamic conditions. Among the most widely adopted techniques, Principal Component Analysis (PCA) plays a crucial role in dimensionality reduction and feature extraction, forming the basis for many subsequent modeling and monitoring techniques.

1.4.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical method used to transform correlated variables into a set of uncorrelated components, called principal components. By projecting data into a lower-dimensional space that preserves the maximum variance, PCA simplifies complex datasets, enhances interpretability, and reduces computational costs. In industrial applications, PCA is frequently applied to identify dominant patterns in multivariate signals and to detect deviations from normal operating conditions, serving as a foundational approach for process monitoring and anomaly detection [15]. The computation of the principal components is achieved through the eigen-decomposition of the data's covariance matrix. The eigenvectors of this matrix represent the principal components, which are the orthogonal axes of maximum variance in the data. The corresponding eigenvalues quantify the amount of variance captured by each component, allowing them to be ranked. The eigen-decomposition is calculated as follows:

$$C_X = \frac{1}{m-1} X^T X \tag{1.1}$$

$$C_X v = \lambda v \tag{1.2}$$

where C_X is the covariance matrix; v is an eigenvector of the covariance matrix; and λ is the corresponding eigenvalue, a scalar that indicates the amount of variance captured by that eigenvector.

1.5 Deep Learning (DL) models

Building upon data-driven principles, Deep Learning (DL) models represent a more advanced class of ML that leverages multi-layered artificial neural networks. Unlike traditional methods that may rely on pre-processed features, DL techniques excel at automatically learning hierarchical representations directly from raw data. This capability is particularly powerful in industrial contexts. This section presents a detailed overview of the DL models utilized in this thesis, emphasizing their roles in prediction and classification for sequential data.

1.5.1 Recurring Neural Network (RNN)

A Recurrent Neural Network (RNN) is a type of Artificial Neural Network (ANN) designed to work with sequential data, where the order of information is important. Unlike standard Neural Networks (NNs), RNNs have a form of memory. They achieve this through a feedback loop, allowing information from previous steps in the sequence to influence the output of the current step. This is computed per time step to determine the current output o_t and the current hidden state h_t. These depend on the current input features x_t and the previous hidden state h_{t-1}. Both o_t and h_t, which are typically vector quantities, allow the network to retain and utilize information from earlier time steps through its hidden state, effectively serving as a form of memory for future predictions.

$$h_t = tanh(\mathbb{W}_h h_{t-1} + \mathbb{W}_x x_t) \tag{1.3}$$

$$o_t = \mathbb{W}_o h_t \tag{1.4}$$

where: $\mathbb{W}_h$, $\mathbb{W}_o$ and $\mathbb{W}_x$ are weight matrices for hidden, output and input states, respectively; $tanh(x) = (e^{2x}-1)/(e^{2x}+1)$ is used for data normalization, returning a predicted vector h_t with elements in the interval $[-1,1]$. RNNs struggle to capture long-term dependencies due to the vanishing and exploding gradient problems that arise in long sequences [16]. These issues cause gradients to either diminish or grow excessively during backpropagation, hindering effective learning.

1.5.2 Long Short-Term Memory (LSTM)

An LSTM network represents an enhanced version of a RNN capable of capturing long-term dependencies through the use of a cell state, in addition to the hidden state employed in standard RNNs. LSTM overcomes the limitation of vanishing gradient [17] through gating mechanisms to control the flow of information and gradients. Figure 1.3 illustrates the LSTM block and its internal computation.

As described in Subsection 1.5.1, the hidden state and cell state correspond to the short-term and long-term memory components, respectively. Each LSTM cell includes three types of gates that regulate the flow of information and determine what should be retained or discarded: (i) the *forget* gate, which decides whether information from previous time steps should be preserved or forgotten; (ii) the *input* gate, which selects the new information to be stored in the cell state; and (iii) the *output gate*, which controls the information used to produce the final output at each time step t. In detail, an LSTM network performs the following operations:

$$f_t = \sigma(\mathbb{W}_{fh} h_{t-1} + \mathbb{W}_{fx} x_t + b_f) \tag{1.5}$$

$$i_t = \sigma(\mathbb{W}_{ih} h_{t-1} + \mathbb{W}_{ix} x_t + b_i) \tag{1.6}$$

$$o_t = \sigma(\mathbb{W}_{oh} h_{t-1} + \mathbb{W}_{ox} x_t + b_o) \tag{1.7}$$

$$\widetilde{C}_t = \tanh(\mathbb{W}_{ch} h_{t-1} + \mathbb{W}_{cx} x_t + b_c) \tag{1.8}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \widetilde{C}_t \tag{1.9}$$

$$h_t = o_t \cdot \tanh(C_t) \tag{1.10}$$

where: $\mathbb{W}_{fh}$, $\mathbb{W}_{ih}$, $\mathbb{W}_{oh}$, $\mathbb{W}_{ch} \in \mathbb{R}^{n_h \times n_h}$ (with n_h representing the dimension of the hidden state vector) and $\mathbb{W}_{fx}$, $\mathbb{W}_{ix}$, $\mathbb{W}_{ox} \in \mathbb{R}^{n_h \times n_x}$ are the weight matrices for forget gate, input gate, output gate, and candidate cell state, respectively (with n_x corresponding to the dimension of the input vector); b_f, b_i, b_o, $b_c \in \mathbb{R}^{n_h \times 1}$ are the bias vectors for forget gate, input gate, output gate, and candidate cell state, respectively; $h_{t-1} \in \mathbb{R}^{n_h \times 1}$ is the previous hidden state (output) of the LSTM cell; $x_t \in \mathbb{R}^{n_x \times 1}$ represents the current input at time t; f_t is the output of the forget gate; i_t corresponds to the output for the input gate; $o_t \in \mathbb{R}^{n_h \times 1}$ is the output of the output gate; $\widetilde{C}_t \in \mathbb{R}^{n_h \times 1}$ represents the state of the candidate cell; $C_t \in \mathbb{R}^{n_h \times 1}$ corresponds to the state of the final cell; $h_t \in \mathbb{R}^{n_h \times 1}$ represents the output of the hidden state, which is the output of the LSTM cell at time t;

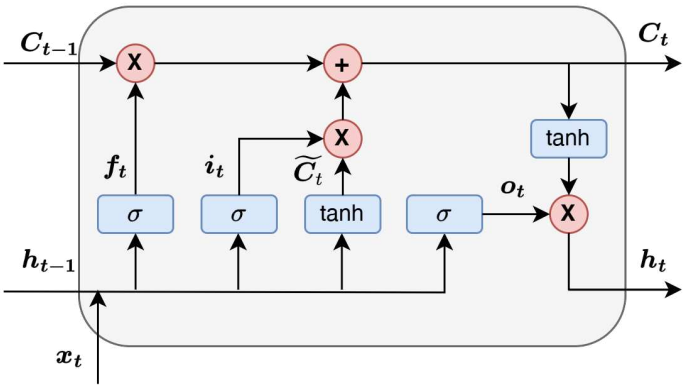


Figure 1.3: Simplified representation of LSTM cell.

1.5.3 Gated Recurrent Unit (GRU)

GRU network can be viewed as a streamlined variant of an LSTM, featuring a simpler architecture by eliminating the separate memory cell and the forget gate. Consequently, a GRU cell relies on just two gates (shown in Figure 1.4): (i) the *update* gate, which controls how much past information from previous time steps is carried forward to the next cell, and (ii) the *reset* gate, which determines the extent to which past information is ignored. Specifically, a GRU network executes the following operations:

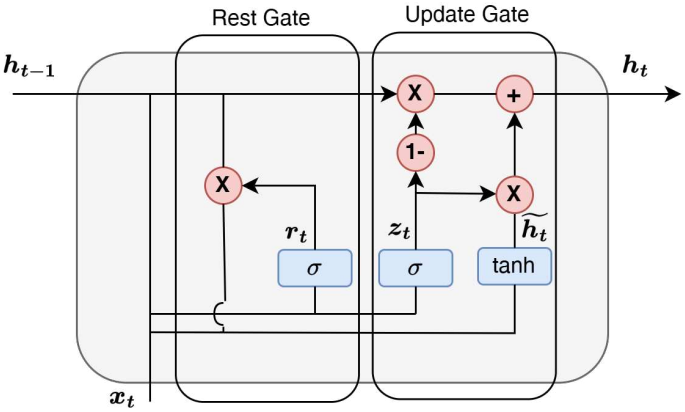


Figure 1.4: Simplified representation of GRU cell.

$$z_t = \sigma(\mathbb{W}_{zh} h_{t-1} + \mathbb{W}_{zx} x_t + b_z) \tag{1.11}$$

$$r_t = \sigma(\mathbb{W}_{rh} h_{t-1} + \mathbb{W}_{rx} x_t + b_r) \tag{1.12}$$

$$\widetilde{h}_t = \tanh(W_{hh}(r_t \odot h_{t-1}) + \mathbb{W}_{hx} x_t + b_h) \tag{1.13}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h}_t \tag{1.14}$$

where: $z_t \in \mathbb{R}^{n_h \times 1}$ is the update gate and $r_t \in \mathbb{R}^{n_h \times 1}$ is the reset gate; $\widetilde{h}_t \in \mathbb{R}^{n_h \times 1}$ is the new candidate activation, with n_h corresponding to the dimension of the hidden state vector; h_t is the new hidden state; $\mathbb{W}_{zh}$, $\mathbb{W}_{rx}$, $\mathbb{W}_{hh} \in \mathbb{R}^{n_h \times n_h}$ and $\mathbb{W}_{zx}$, $\mathbb{W}_{rx}$, $\mathbb{W}_{hx} \in \mathbb{R}^{n_h \times n_x}$ are the weight matrices (with n_x representing the dimension of the input vector); b_z, $b_r \in \mathbb{R}^{n_h \times 1}$ are the bias vectors; $\odot$ corresponds to the element-wise multiplication.

1.5.4 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are primarily used for processing and analyzing visual data, such as images and videos. Its architecture is inspired by the human visual cortex, and it's

particularly effective at identifying patterns and features within data. The core component of a CNN is the convolutional layer, which uses filters (or kernels) to scan over the input data and create feature maps. These feature maps highlight specific characteristics, such as edges, corners, and textures. CNNs also typically include pooling layers to reduce the dimensionality of the data and make the model more efficient, as well as fully connected layers at the end to perform classification or regression tasks. In particular, one-dimensional (1D)-CNNs are commonly employed for time-series data, as illustrated in Figure 1.5.

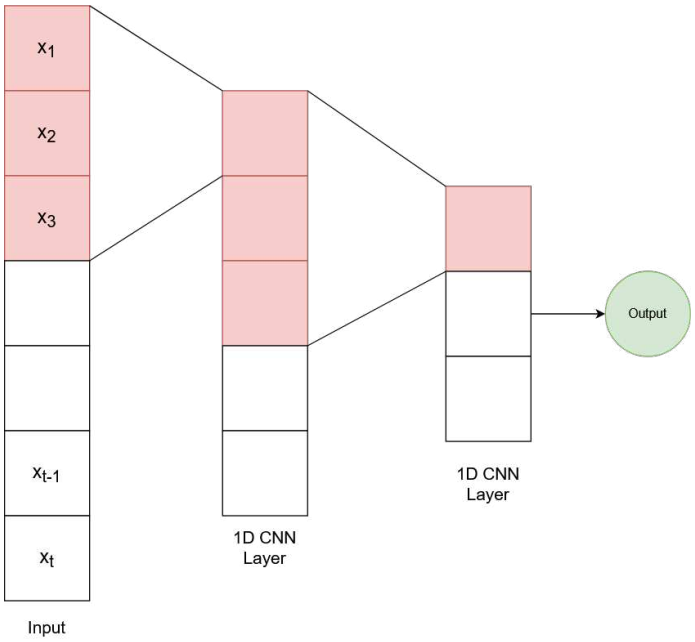


Figure 1.5: Simplified representation of CNN architecture.

1.5.5 Temporal Convolutional Network (TCN)

A Temporal Convolutional Network (TCN) is a specialized form of 1D-CNN designed for effective analysis of sequential data [18]. The key features that make TCNs powerful are:

- Causal Convolutions: This is a crucial property ensuring that the prediction at a given time step t can only depend on inputs from t and earlier. It prevents the model from using future information, which is essential for tasks like forecasting.
- Dilated Convolutions: To see further back in the past without a massive increase in computational cost, TCN layers use dilated convolutions. This technique involves applying a filter over an area larger than its length by skipping input values with a certain step. The dilation factor is typically increased exponentially with each layer, allowing the network's receptive field to grow very quickly.

Figure 1.6 illustrates the differences between standard convolution, causal convolution, and dilated convolution.

1.6 Challenges and Research Gaps

Despite the remarkable progress achieved by data-driven and DL techniques, several challenges still hinder their widespread adoption in industrial settings. One of the primary issues concerns the inherent scarcity of high-quality data for anomalous events, leading to highly imbalanced datasets that complicate the training of robust and accurate detection models. Additionally, the

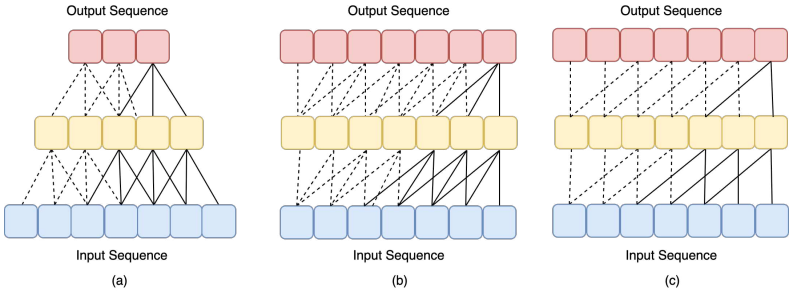


Figure 1.6: (a) Typical convolutional network with kernel size equal to 3; (b) causal convolutional network with kernel size equal to 3; (c) dilated causal convolutional network with kernel size equal to 3 and dilation rate equal to 2.

interpretability and transparency of DL models remain open problems, particularly in safety-critical domains such as pharmaceutical or chemical manufacturing, where model decisions must be explainable and trustworthy.

Another critical challenge in industrial IoT environments is the trade-off between model complexity and computational efficiency. While large and deep architectures can effectively capture non-linear dynamics and temporal dependencies, their high computational cost makes them difficult to deploy on resource-constrained devices commonly used in industrial IoT applications. Achieving accurate, interpretable, and computationally efficient models that can be deployed on real-time, low-power devices therefore represents a key research frontier.

1.7 Research Objectives

This thesis aims to bridge the gap between high-performance AI models and their real-world application, also in resource-constrained environments. The research systematically explores, develops, and evaluates data-driven frameworks for monitoring, prediction, and security, with a consistent focus on the critical trade-off between model accuracy and computational efficiency. The primary objectives are:

- To design and validate effective data-driven methodologies for fault and anomaly detection in complex industrial processes, specifically within the pharmaceutical field.
- To conduct a comprehensive, comparative analysis of state-of-the-art DL architectures for time-series forecasting and classification tasks relevant to environmental monitoring and network security.
- To systematically investigate the impact of model optimization and compression techniques—including PTQ and WP—on the performance and footprint of DL models.
- To establish a clear framework for evaluating the deployability of these models on constrained hardware, balancing predictive or detective accuracy against computational complexity, memory usage, and execution time.

The contributions of this thesis range from the development of models for industrial processes—where optimizing complex models is as important as analyzing large amounts of data—to the deployment of models tailored for edge computing, using techniques to lighten the models while maintaining high accuracy.

1.8 Common Features of the Proposed Studies

Although the research presented in this thesis addresses diverse application domains, it is underpinned by a unified methodological approach. The common features that weave these studies into a cohesive body of work can be identified as follows:

- **Multivariate Time-Series Analysis:** All chapters deal with high-dimensional data characterized by complex temporal dependencies and non-linear dynamics. Whether analyzing sensor data from a chemical reactor or traffic patterns in a network, the core challenge remains the extraction of meaningful patterns from noisy, real-time data streams.
- **Deep Learning as a Core Engine:** The research consistently leverages advanced neural architectures—specifically LSTM, GRU, and TCN—to model the underlying behavior of the systems under study. Models are selected based on their specialized ability to capture complex temporal dependencies and non-linear patterns within high-dimensional data streams, ensuring a high degree of representational power across several scenarios.
- **Focus on Practical Deployability:** A defining theme of this research is the transition from theoretical models to operationally viable solutions. Each study moves beyond mere predictive accuracy to rigorously evaluate the computational footprint of the architectures. By explicitly addressing the hardware constraints of resource-limited edge devices, the research ensures that the proposed models are optimized for real-time execution in actual autonomous systems environments.
- **Multi-Objective Evaluation:** The studies share a common evaluation paradigm that balances predictive performance with architectural efficiency. This is exemplified by using performance scores, which can be converted to show a trade-off between accuracy and hardware resource consumption in edge scenarios, providing a consistent metric for cross-domain model assessment.

By integrating these features, the thesis provides a comprehensive framework for the design of the next generation of intelligent, autonomous, and efficient monitoring systems.

1.9 Thesis Structure

Guided by the common features and methodological framework outlined in Section 1.8, the remainder of this thesis is organized into two main parts that together illustrate the progressive development of data-driven monitoring and optimization frameworks, moving from industrial process diagnostics to edge-oriented cybersecurity applications.

The first part focuses on the development and refinement of anomaly detection and predictive monitoring methodologies within the pharmaceutical manufacturing domain.

Chapter 2 introduces an analysis of leak detection cycles in pharmaceutical lyophilizers. The study applies Principal Component Analysis (PCA) for dimensionality reduction and explores multiple clustering algorithms to identify anomalies and system degradation. The outcome of this work is the construction of a quantitative Health Indicator (HI) that enables systematic assessment of equipment conditions.

Chapter 3 extends the industrial application by presenting a predictive monitoring framework for pharmaceutical batch processes. Building on the previous chapter, it employs an Long Short-Term Memory (LSTM)-based Auto Encoder (AE) to model the temporal evolution of the process and integrates an adaptive rolling-threshold mechanism for real-time and sensitive anomaly detection.

The second part transitions toward the deployment and optimization of ML models in resource-constrained and security-critical environments.

Chapter 4 explores model efficiency in an air quality prediction task, using it as a representative case for Internet of Things (IoT) scenarios. It compares a broad spectrum of Machine Learning (ML) and Deep Learning (DL) models, offering the first comprehensive evaluation of Post-Training Quantization (PTQ) techniques to drastically reduce model size while maintaining prediction accuracy [19].

Chapter 5 applies the optimized frameworks to the field of cybersecurity, specifically focusing on network intrusion detection for IoT systems. It systematically evaluates multiple deep learning architectures, quantifying their computational complexity and exploring the accuracy–efficiency trade-off critical for edge deployment.

Chapter 6 deepens this cybersecurity focus within Software-Defined Networking (SDN) environments. It compares a diverse suite of DL models and investigates several compression and

quantization techniques, including Weight Pruning (WP). This Chapter introduces the Normalized Composite Accuracy-Size (NCAS), a novel metric that quantitatively balances model size and performance, offering actionable insights for selecting deployable intrusion detection systems.

Collectively, this thesis provides a comprehensive roadmap—from foundational data analysis to advanced DL and model compression—demonstrating the feasibility and effectiveness of deploying optimized AI-powered monitoring and security solutions also at the network edge. This research provides a multi-faceted contribution to the field of intelligent autonomous systems. It demonstrates the successful application of advanced AI from industrial process control to network security, with a unique and consistent emphasis on the practical challenges of edge deployment. The findings presented herein offer valuable insights for researchers and practitioners aiming to develop the next generation of efficient, reliable, and secure monitoring systems.

Chapter 2

Data analysis of leak detection cycles in pharmaceutical lyophilizers

2.1 Introduction

2.1.1 Freeze-drying process

Freeze-drying is a process that involves three different phases which eliminates water from the product via sublimation. Firstly, in the *freezing* phase, the product is cooled below its triple point, converting water into ice. Subsequently, during *primary drying* phase, chamber pressure is reduced below the vapor pressure of ice at the product's temperature, causing sublimation of the ice. Finally, in *secondary drying* phase, temperature is gradually increased to remove residual moisture from the product, thereby eliminating unfrozen water molecules that are chained with product molecules. Since the process deals with high variations in both pressure and temperature, the lyophilizer is subject to thermal fatigue, possibly leading to microscopic cracks, which ultimately result in leaks, allowing gas to infiltrate in the drying chamber and impact the whole process. For a comprehensive explanation of freeze-drying, refer to [20].

2.1.2 Leaks

A leak in a freeze-dryer refers to the unintended influx of gas or vapor into the system, compromising the efficiency and effectiveness of the drying process. This leads to incomplete drying or contamination of the product, depending on the location of leaks.

Specifically for freeze-dryer, leaks can occur in various parts of the system, including seals, valves, connections, or the chamber itself. Normally, leaks impact increase over time due to the expansion of holes or cracks. Detecting and repairing leaks is crucial to maintain the integrity of the freeze-drying process and ensure product quality.

2.1.3 Leak Detection Test

Typically, a leak detection test is executed before freeze-drying to assess leak rate [21] and monitor potential gas infiltration. The leak detection test is conducted prior to commencing the lyophilization process. During this test, the trend of increasing pressure over time is observed. The rate of pressure is then evaluated and, if exceeds a predetermined threshold, it indicates improper sealing of the chamber, resulting in production loss due to maintenance. Typically, a leak test comprises of three distinct sub-phases:

- Evacuation: The chamber pressure decreases due to the action of a pump creating a vacuum.
- Pre-leak test: Pressure is sustained in a vacuum state for a specified duration. Usually, if the pressure exceeds a pre-defined threshold more than three times, we may prematurely conclude the leak test, thus assuming leaks in the system. However, in the targeted freeze-dryer, vacuum pumps are activated if pressure exceeds the threshold and maintain the level in vacuum range.
- Leak test: Any action from vacuum pumps are neglected and the freeze-dryer is kept in this state for a fixed duration, defined by the freeze-dryer specification.

Nevertheless, successfully passing the leak test does not guarantee the absence of all leaks.

This study provides a model to decompose subsequent cycles data into a reduced data space, while retaining meaningful information. The decomposition is then used as input for clustering methods, identifying anomaly or deteriorated cycles. Once the cycles are accurately identified, we calculate an Health Indicator (HI) to evaluate the overall leak impact.

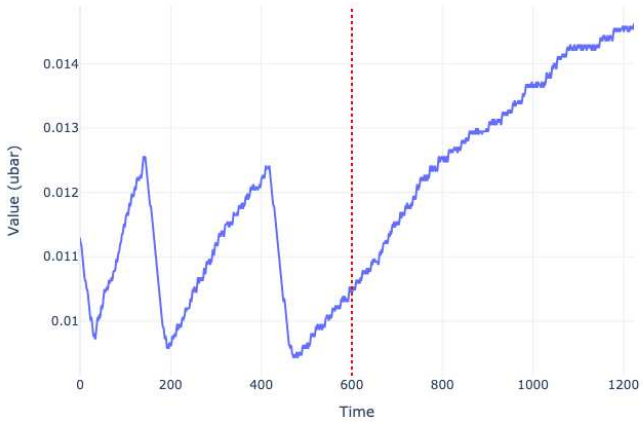


Figure 2.1: A typical Leak Test cycle. The red vertical line separates the pre-leak test phase from the leak test phase.

Research Outline

In Section 2.2, we propose related freeze-drying model approaches to model phases dynamics or identify potential anomalies. In Section 2.3, we define the data pipeline, from preprocessing techniques to models architecture, showing each step output using real leak test data from GlaxoSmithKlaine (GSK)'s production plant in San Polo di Torrile (Parma, Italy). Moreover, in Section 2.4 we define the HI and some Machine Learning (ML) models to predict future cycles. Section 2.5 concludes the study with a clear review of the models and future implementations.

2.2 Literature Review

This study aims to present a methodology for anomaly detection that can be applied to any system analysis, with the flexibility for tuning as needed. Specifically to our environment, there is limited research regarding the extraction of information from leak tests. Given that each phase of the manufacturing process can exhibit abnormal behavior, conducting a thorough analysis of significant phases may reveal underlying patterns that contribute to faults. In [22], clustering methodologies are employed to evaluate Health Indicators (HIs) by analyzing data from the Clean In Place (CIP) phase. This predictive approach proves effective with limited data and features by first aggregating cycle information and then applying clustering techniques. Initially, the HI is derived by detecting multiple clusters from aggregated cycles, illustrating a trend in water flow rate. Subsequently, a signal is generated using the centroid of each cluster, calculating the distance of data points from their respective cluster centroids. This signal demonstrates a temporal trend and accurately coincides with maintenance activities, indicating a drop during those periods.

In [23], an identification-based approach is developed to evaluate two categories of leaks by analyzing both the pre-leak test and leak test phase. This is achieved with an optimization framework, where equations representing physical interactions are incorporated as constraints. The model effectively discerns between external leaks, which remain constant over time, and internal leaks, characterized by exponential decay terms. As a result, it accurately estimates the pressure during the ongoing leak test phase as the sum of leak contributes, thereby evaluating leak dynamics and identify cycles that are prone to faults by setting thresholds.

One of the issue on analyzing manufacturing process data is the lack of good amount of faults or "bad" cycle, which leads to unbalanced dataset, limiting the choice of modeling. For such that reason, the majority of freeze-dryer modeling rely on physical modeling of meaningful phases such as freezing and primary drying. In [24], a Quality by Design (QbD) framework is employed and used to estimate the design space concerning Critical Quality Attributess (CQAs) of the product and Critical Process Parameters (CPPs). The QbD approach ensures the attainment of

the desired state for the final product at the end of the manufacturing process, with a particular emphasis on product alterations throughout the process. As QbD modeling centers on product quality, it integrates risk assessment and management across the product lifecycle. This involves the identification of potential risks to product quality, their prioritization based on impact, and the implementation of strategies to mitigate or eliminate these risks.

In [25], a physical model for off-line optimization and on-line monitoring of primary drying phase is developed by assuming pseudo-stationary conditions. By making appropriate simplifications, it becomes feasible to approximate CQAs and CPPs in slow dynamics systems. Acquiring nearly optimized parameters, including heat and mass transfer, leads to a reduction in phase duration while maintaining optimal temperature and pressure, thereby achieving the desired product state faster.

However, numerous studies utilize Principal Component Analysis (PCA) for data analysis purposes. The main goal of PCA is to simplify the complexity in high-dimensional data while retaining trends and patterns. In [26], PCA and Partial Least Squares (PLS) methodologies are utilized for spectral analysis to assess the residual phenol content in both spray-dried and freeze-dried insulin formulations. Fourier Transform InfraRed (FTIR) and Near InfraRed (NIR) spectra were compared with the phenol/insulin ratio using PLS and PCA analysis. Notably, the spray-dried samples exhibited higher water content and lower phenol levels, while the freeze-dried counterparts showed lower water content and higher phenol levels. For both spectra, PLS reports better description of phenol/insulin ratio in the powders, carefully tuning the number of components to represent the majority of the variance.

2.3 Proposed Method

In this Section we provide a concise overview of the data processing and cleaning techniques employed. Additionally, we discuss the utilization of PCA for subspace projection, and a comparison between clustering techniques is made to correctly detect anomalies.

2.3.1 Data Preprocessing

The raw pressure signal is illustrated in Figure 2.1. Generally, the acquisition of these data represents two fundamental issues:

- Signals are noisy and may impact the data analysis, thus obtaining inconsistent processed anomalies. We can achieve smoothness by means of moving average technique, improving the detection of underlying trends or patterns.
- Data gathered from sensors are selectively stored in the database, occurring only when a significant variation from the previous value is detected. Therefore, interpolation becomes necessary to ensure uniform sampling intervals and obtain signals with consistent time sampling;

2.3.1.1 Signal Interpolation

Given that the signal of interest is delineated during the leak detection test, we exclusively focus on that specific segment. We analyze the signal beyond the red vertical line shown in Figure 2.1. However, it's important to acknowledge that including the pre-leak test portion of the signal can provide additional valuable insights for alternative applications, as demonstrated in [23]. Nevertheless, this approach may be useless as most newer freeze-dryers no longer rely on vacuum pumps to maintain chamber pressure stability, therefore we can generalize this approach for almost any leak detection tests.

The linear interpolation computes new data points as follow:

$$y = y_0 + \frac{(x - x_0) \cdot (y_1 - y_0)}{x_1 - x_0} \tag{2.1}$$

where (x_0, y_0) and (x_1, y_1) are known subsequent data points.

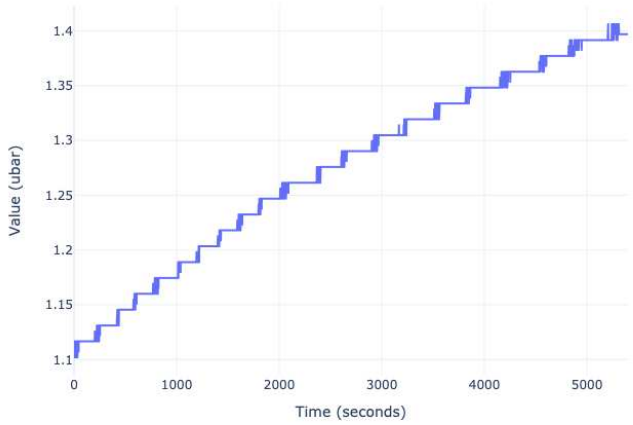


Figure 2.2: Leak test pressure signal obtained through interpolation between data points.

2.3.1.2 Signal Smoothing

One commonly used method for signal smoothing is the moving average filter. The moving average filter is a simple yet effective method for smoothing signals. It operates by replacing each data point with the average of nearby data points within a specified window or interval. This averaging process effectively filters out high-frequency noise while preserving the general shape and trend of the signal.

Given a signal x of length N, and a window size M, each k value of the smoothed signal y using a moving average filter is calculated as:

$$y(k) = \frac{1}{M} \sum_{j=k-\frac{M-1}{2}}^{k+\frac{M-1}{2}} x(j), k = 1, \ldots, N \tag{2.2}$$

Increasing the value of M results in a signal that is more smoothed, but at the cost of losing information.

The resulting signal, shown in Figure 2.3, is obtained using a window size equal to 50 and interpolating the raw pressure signal, thus resulting in a signal with sampling rate equal to 1 minute. The signal lasts for a total of 90 minutes, defined as default time for our freeze-dryer specifications.

2.3.1.3 Data scaling

Scaling ensures that all variables have the same influence on the Principal Components (PCs) of PCA, regardless of their original scales. This is important because PCA is sensitive to the relative scales of the variables. For this purpose, we scale the data by removing the mean and dividing to unit variance:

$$y = \frac{x - \mu}{\sigma} \tag{2.3}$$

where x is the input signal; μ is the mean of the signal; and σ is variance of the signal.

2.3.2 Principal Component Analysis (PCA)

As detailed in Section 1.4, PCA is also widely employed in data analysis for reducing data dimension. Its core objective is to project high-dimensional data into a low-dimensional subspaces, referred to as PCs, wherein the variability of the data is maximized along the axes. This leads to a linear transformation of the data onto a new coordinate system, enabling the

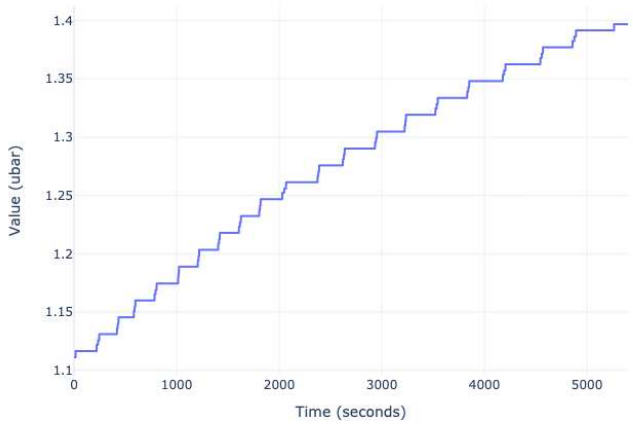


Figure 2.3: Processed leak test signal obtained through interpolation and smoothing.

straightforward identification of directions that capture the most significant variation in the dataset.

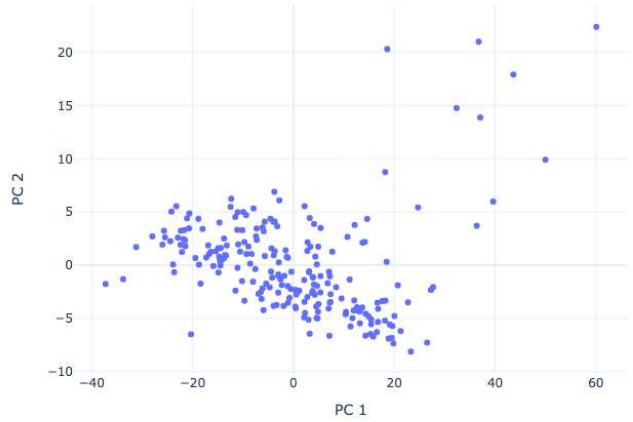


Figure 2.4: Projection of the first two components, underlying a clear identification of good cycles.

PCs are obtained through linear combinations of the original variables [27]. Given a matrix *nxm*, with *n* samples and *m* variables, we can define the PCA dataset as follow:

$$X = TP^t + E = \hat{X} + E \tag{2.4}$$

where T is the *score* matrix of size *nxr*; P is the *loading* matrix of size *mxr*; and E is the residual matrix, which has the same size of the original dataset. The number of *r* components is selected in order to retrieve the most possible variance, while preserving information. As shown in Figure 2.5, selecting the first 2 components in our dataset, we can achieve more than 99% of variance caught.

2.3.3 Clustering approach

Since the obtained dataset from PCA is suitable for general machine learning or data analysis approaches, the resulting dataset can be employed in clustering and outlier identification algorithms. For this purpose, numerous methods can yield accurate results, therefore a comparison among several general clustering algorithms is essential to obtain the optimal outcome.

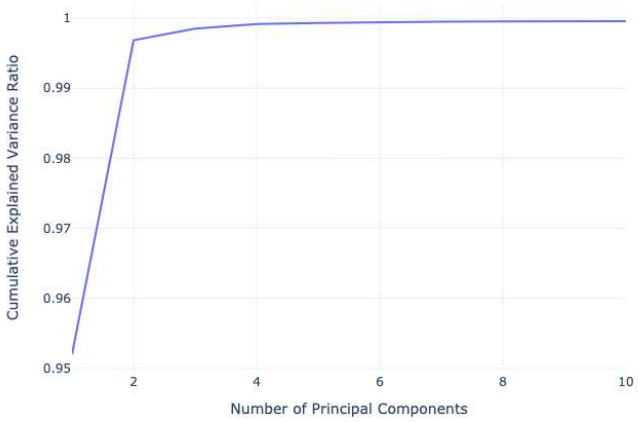


Figure 2.5: Cumulative Explained Variance Ratio based on number of components.

2.3.3.1 K-means clustering

K-means clustering is a fundamental unsupervised machine learning technique used to group data points into clusters based on their similarities. By specifying the desired number of clusters for the dataset, the algorithm identifies the most suitable centroids within the data space. This is achieved by randomly selecting k points and calculating the distance from these centroids. The outcome is K clusters, each characterized by its centroid, with data points allocated to the cluster whose centroid they are closest to. Since the number of clusters is arbitrary, the value is highly dependant on the dataset. Opting for a small number of clusters limits our ability to accurately identify outliers. We can compute the optimal K clusters employing the elbow method. This method relies on a visual examination to determine the Within-Cluster Sum of Squares (WCSS), which is the sum of the squared distances between points in a cluster and the cluster centroid. It involves iteratively running the K-Means algorithm with varying numbers of clusters K until convergence is no longer achieved. At this point, the WCSS value for each cluster set is obtained. By analyzing this representation, the "elbow" of the curve is identified, indicating the optimal number of clusters K.

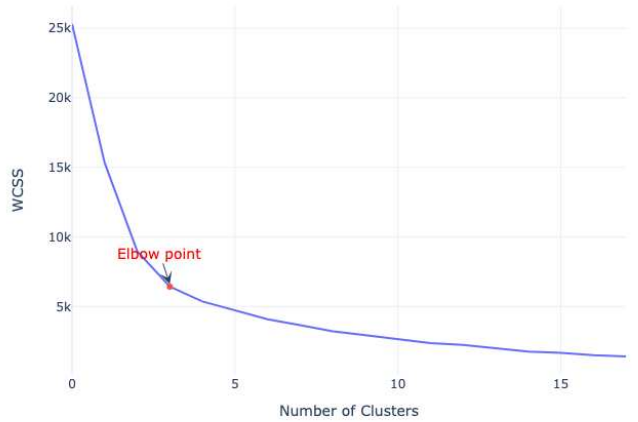


Figure 2.6: Optimal number of clusters evaluation through elbow method.

As defined in Figure 2.6, the optimal number of clusters K for our dataset is equal to 3.

Running K-Means Clustering with 3 clusters (see Figure 2.7), we can correctly distinguish any cycle deterioration over time.

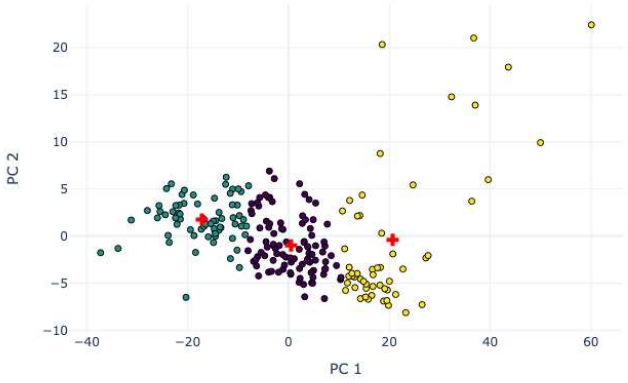


Figure 2.7: K-means clustering selecting 3 clusters.

2.3.3.2 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN is particularly useful for identifying clusters of varying shapes and sizes in datasets containing noise and outliers. It achieves that by grouping together closely packed points in the dataset based on their density. It defines clusters as dense regions separated by sparser areas. As conducted in [22], DBSCAN algorithm is efficient and accurate in detecting outliers in this scenario. Moreover, DBSCAN parameters can be tweaked in order to optimize the clustering detection. The algorithm requires the estimation of two parameters: (i) the distance ε represents the radius of a neighborhood surrounding a particular point within the cluster, and (ii) the minimum number of items per cluster. To compute these parameters, we can apply a similar approach of K-Means Clustering based on elbow evaluation.

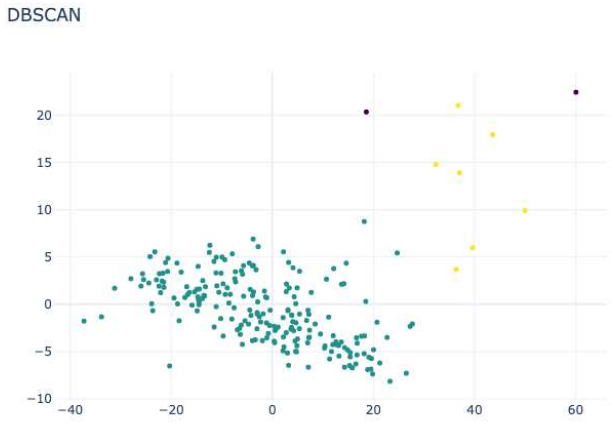


Figure 2.8: DBSCAN clustering with the set (epsilon, min_samples) of parameters.

2.3.3.3 Isolation Forest (IF)

IF is an unsupervised machine learning algorithm used for anomaly detection. The algorithm works by isolating anomalies in the dataset by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of that feature. This process is repeated recursively until all data points are isolated or until the maximum number of isolation steps is reached. Anomalies, being different from the majority of the data points, are expected to be isolated in fewer steps compared to normal data points. The main challenge with IF is correctly defining the contamination parameter. This parameter helps the algorithm in determining the number of anomalies to be detected. By setting the contamination parameter appropriately, the algorithm can adjust its threshold for identifying anomalies. For example, if you expect that 1% of your data points are anomalies, you might set the contamination parameter to 0.01. Hence, the algorithm assumes prior knowledge regarding the number of outliers, a factor that may not always be known in practice.

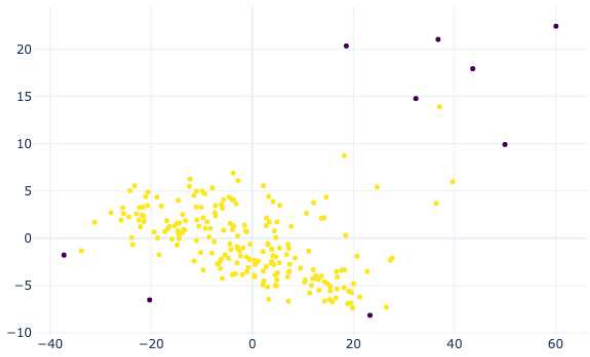


Figure 2.9: IF algorithm with contamination value of 0.04. The algorithm detects some "good" cycles as outliers.

Upon examining clustering outcomes, IF may be able to identify anomalies with prior knowledge, but proves inadequate for detecting deterioration. Conversely, DBSCAN effectively identifies outliers and potential deteriorated cycles, whereas K-Means clustering may group these cycles alongside ostensibly "good" ones, potentially identify anomalies or deterioration before they occur. Consequently, we opted for K-Means Clustering as the preferred method for accurately analyzing leak test cycles, enabling the retrieval of signals indicating deterioration or anomalies.

2.4 Health Indicator (HI)

HIs generally entail grouping similar data points based on different attributes or variables. The objective of these indicators is to discern patterns or clusters within a population or data points, which could signify varying levels of health status. This approach proves especially beneficial in evaluating trends or irregularities in manufacturing systems, where monitoring is crucial for determining the overall effectiveness of the process. In our case, to calculate the HI for leak tests, we compute the Euclidean distance of each i-th data point from its respective cluster centroids.

$$\mathrm{d}[i] = \sqrt{\sum_{j=1}^{n} (X_{\mathrm{pca}}[i,j] - \mathrm{centroids}[i,j])^2} \tag{2.5}$$

Where: n represents the number of dimensions, i.e., number of PCs; $X_{\mathrm{pca}}[i,j]$ is the j-th coordinate of the i-th data point in PCA-transformed space; and $\mathrm{centroids}[i,j]$ is the j-th coordinate of the centroid of the cluster to which the i-th data point belongs.

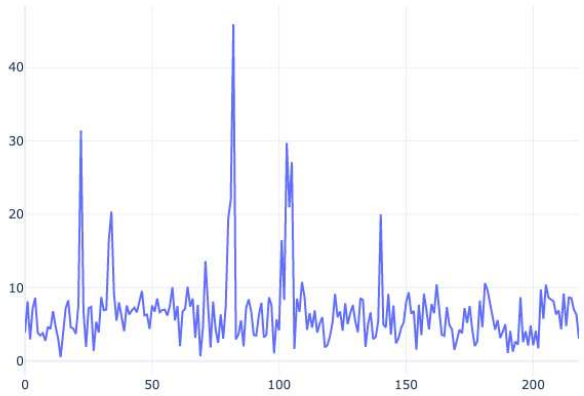


Figure 2.10: HI computed with the euclidean distance of each data point from its relative cluster centroid.

Finally, the resulting signal is smoothed using Savitzky-Golay (SavGol) filter to retain the precision of the data without impacting its tendency. The SavGol filter is a digital filter used for smoothing noisy data. The filter works by fitting a polynomial to a small window of adjacent data points and then using the coefficients of this polynomial to compute a smoothed value for the central data point in the window. This process is repeated for each data point in the dataset, effectively smoothing out the noise while preserving the underlying trends in the data.

$$\hat{y}_i = \sum_{j=0}^{N} c_j y_{i+j} \tag{2.6}$$

Where: $\hat{y}_i$ is the smoothed value of the data point at index i; y_{i+j} are the data points within the window centered at index i; N is the degree of the polynomial used for fitting; and c_j are the coefficients of the polynomial. The key advantage of the SavGol filter is that it provides a good compromise between noise reduction and preservation of important features in the data.

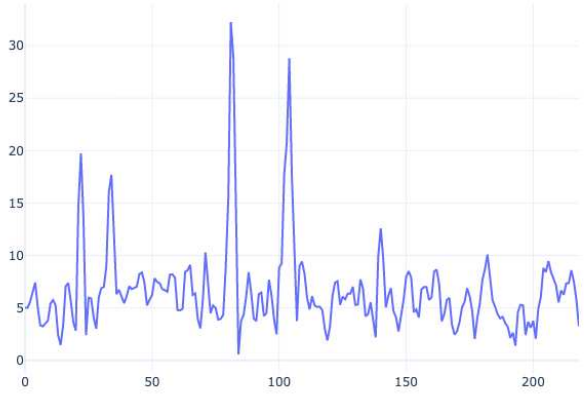


Figure 2.11: Smoothed HI using a second order polynomial SavGol filter with window size equal to 5.

The resultant signal underscores cycles with a substantial leak impact, facilitating an accurate interpretation of leak dynamics.

2.5 Conclusions

In this study, we reviewed the leak problem in freeze-dryers and leak test phase, outlining possible studies on that particular phase. Moreover, a subspace approach is used to obtain a low-dimensional dataset, finding deep correlations and patterns among variables. Clustering methods are then performed to quickly identify anomalies or deteriorated cycles. As a result, K-Means Clustering have found the best suitable clusters to identify them, and an HI is computed from cluster centroids to predict future cycles. The presented data analysis can be employed in different phases of the manufacturing process by carefully selecting the most important features to project in low-dimensional spaces. Moreover, the computed HI may be a valid input for predictive techniques such as ML methods or Neural Network (NN) models.

Chapter 3

Enhancing Pharmaceutical Batch Processes Monitoring with Predictive LSTM-based Framework

3.1 Introduction

An anomaly is an unexpected deviation from normal system behavior, representing data points or events that stray from the operational baseline. These anomalies can indicate critical issues—such as faults, errors, or fraudulent activities—that may lead to degraded performance, failures, safety risks, or financial losses. Pharmaceutical processes, in particular, are complex and strictly regulated. Anomalies in these settings can result from equipment malfunctions, environmental changes, human errors, or variations in raw material properties, making swift detection essential for maintaining process integrity. However, detecting anomalies is particularly challenging in batch operations, which involve dynamic, stage-specific behaviors and batch-to-batch variability (see [28]). An anomaly in one stage might be typical in another, further complicating detection. Additionally, batch-to-batch variability—driven by factors like raw material quality, environmental conditions, or equipment wear—further obscures the identification of subtle anomalies [29].

Over the past decade, various data-driven methods have been employed for anomaly prediction. Traditional statistical techniques, such as PCA [30] and PLS [31], have been widely used, even though their linear nature limits their ability to capture non-linear interactions in batch data. Extensions like kPCA [32] and non-linear PLS have been developed but often come with high computational costs. Recent advances in ML and DL—particularly Auto Encoder (AE) and Recurrent Neural Network (RNN) [33]—have shown promise in capturing both non-linear patterns and temporal dependencies in dynamic systems.

This research introduces a robust framework that combines Long Short-Term Memory (LSTM) and AE methodologies (detailed in Section 3.3) for batch process monitoring. It demonstrates improved performance in managing the complex dynamics of batch process monitoring in comparison to traditional approaches (discussed in Section 3.4.1). Furthermore, a customized threshold mechanism, based on rolling median and Mean Absolute Deviation (MAD) is implemented to enhance anomaly detection accuracy and reduce false positives. Notably, the framework also employs an LSTM regression model to predict future process variables, which are subsequently fed into the AE to enable predictive anomaly detection.

Research Outline

In Section 3.2, we discuss existing approaches for anomaly detection in batch processes, including statistical and ML-based methods, and we outline the motivation behind our proposed framework. Section 3.3 introduces the architecture, implementation, and integration of our framework composed by an LSTM-AE model alongside the LSTM regression model, designed for both real-time monitoring and prediction. The training and testing of these models utilize batch process data sourced from the pharmaceutical company GlaxoSmithKlaine (GSK) (as detailed in Section 3.3.1). Section 3.4 describes the experimental setup, evaluation metrics, and provides a comparative analysis of our approach against alternative methods. Finally, Section 3.5 concludes the research by summarizing key findings, discussing their implications, and suggesting future research.

3.2 Literature Review

Despite the inherent non-linearity of industrial processes, PCA remains popular for process modeling due to its simplicity and ease of use [34]. An alternative for batch processes, as shown

in [35], employs a multi-way PCA technique by aligning and concatenating batches into an unfolded 2D matrix, suited for following transformations. The advantages of PCA in industrial applications include robust irregularity detection even with sparse data, scalability for process efficiency, and ease of interpretation for real-time monitoring and control.

Transitioning from linear to non-linear methods introduces challenges in deploying optimized, interpretable models for fault detection in highly non-linear systems. Non-linear techniques offer enhanced modeling capabilities but often demand greater computational resources and larger training datasets. For instance, kernel PCA—presented in [36]—addresses linear PCA's limitations by modeling complex non-linear relationships. However, kPCA requires significant computational resources due to the need to compute and store a kernel matrix that scales quadratically with the number of samples, and its high-dimensional feature space can complicate interpretation.

Building on these non-linear techniques, methods such as one-class Support Vector Machine (SVM) [37] and Support Vector Data Description (SVDD) [38] have been adopted to enhance anomaly detection. Both these methods use kernel functions to map data into a high-dimensional space based solely on fault-free samples, with a key distinction in the boundary: one-class SVM constructs a hyperplane while SVDD defines a hypersphere. In [39], one-class SVM is compared with Deep Neural Networks (DNNs) for detecting anomalies in the context of a water treatment plant. The study demonstrates that while both methods can be effective, each has its own trade-offs in terms of false positives and sensitivity to different fault scenarios. In [40], a deep variant of SVDD is developed to detect anomalies in industrial machinery based on audio signals. By mapping log-Mel spectrograms into a feature space and learning a compact hypersphere that encloses normal behavior, the method achieves excellent detection performance under various noise conditions. These studies not only demonstrate how one-class SVM and SVDD can effectively model normal operational states and identify deviations as anomalies across various types of industrial data, but they also pave the way for the adoption of kernel-based and DL approaches in anomaly detection and dimensionality reduction tasks.

In the past decade, Auto Encoders (AEs) have emerged as one of the most effective methods for anomaly detection in non-linear systems due to their ability to learn compact and expressive representations of complex data [41]. By reconstructing the underlying data distribution through a decoder, AE-based models can inherently detect deviations from learned patterns—a feature particularly valuable in batch operations. Studies such as [42] and [43] have demonstrated the potential of AE-based approaches to enhance fault detection and process monitoring.

Statement of Contribution

AEs serve two main functions: (I) reconstruction-based detection and (II) prediction [44]. In this work, we introduce a framework that combines an LSTM-AE for reconstruction-based detection with a separate LSTM regression model for variable prediction. The models are trained on a batched dataset with variable batch lengths, reorganized into a two-dimensional matrix that preserves the temporal structure without using interpolation or padding. This approach tackles the challenges posed by non-linear dynamics, high dimensionality, and temporal dependencies in batch data. Although metrics such as Hotelling T^2 and Squared Prediction Error (SPE) are used for outlier detection in linear models [45], Hotelling T^2 is rarely applied with AEs because its assumptions—linear relationships, Gaussian latent distributions, and a well-defined covariance matrix—do not hold in DNNs. Therefore, we utilize a non-parametric threshold computed using a rolling median and rolling MAD for more reliable anomaly detection.

Overall, our framework is capable of: (I) achieving robust reconstruction by emphasizing key features while filtering out noise; (II) implementing an anomaly detection mechanism that improve the detection of anomalies and deterioration over time and reduces false positives; and (III) capturing temporal dependencies and long-term patterns by predicting process variables.

3.3 Proposed Method

This section presents our framework for anomaly detection and prediction in batch processes, detailing the data preprocessing steps and overall data flow (see Figure 3.1). The framework starts

with raw data, which is initially standardized using a scaling method. The data then proceeds through two training phases: one for the LSTM regression model and another for the LSTM-AE. For the regression model, data is formatted with past and future steps to accurately forecast sequences of points. For the AE, the reconstructed output is subsequently evaluated against a threshold to detect anomalies.

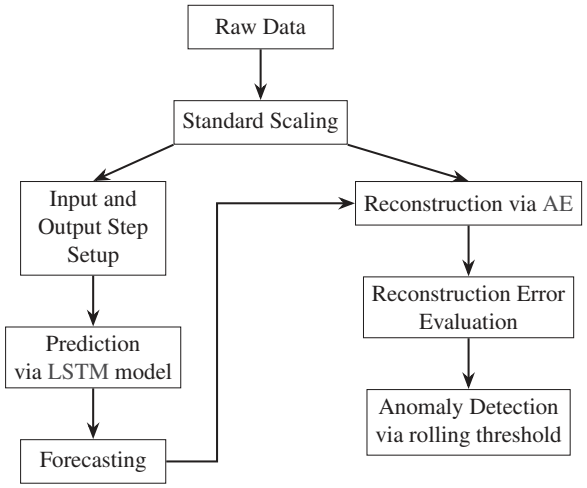


Figure 3.1: Data flow of both prediction and reconstruction phases.

3.3.1 Data Architecture

The dataset for this study comes from a GSK manufacturing process that produces proteins for a biologic drug. In this process, protein cells are cultivated in large bioreactors under precisely controlled conditions—such as pressure, pH, and oxygen levels—to ensure optimal cell growth. Once the cells produce sufficient proteins, they are extracted, purified through a series of filtration and chromatography steps to eliminate impurities, and then rigorously tested for quality and safety. Each variable must operate within its designated Normal Operating Condition (NOC) to maintain the process's integrity. A preliminary variable selection, conducted by the company, has identified critical process variables—referred to as dynamic variables—for monitoring. These include measurements from pH sensor and controller, Dissolved Oxygen sensor and controller, Vessel Weight, Vessel Pressure, and levels of Air, CO_2, O_2, and N_2, making a total of 10 key variables. Given the multivariate analysis and prediction, the pH signal (shown in Figure 3.2) was chosen as the reference signal since it exhibits the highest variance among all signals.

The data is structured in batches, with each batch representing a distinct production cycle and capturing time series data from multiple sensors, with sampling period equal to 5 minutes. Although each batch follows a similar pattern, subtle variations exist, requiring the model to detect anomalies by discerning these differences. The dataset comprises 14 training batches and 6 test batches, all adhering to the process's NOC. As noted earlier, each batch terminates upon reaching the required protein quantity, resulting in variable batch lengths. Aligning these batches is challenging, often requiring the introduction of artificial sequences of points to standardize the lengths. However, this process can distort the original data distribution, introducing more noise or misleading patterns that may compromise the model's ability to learn meaningful representations and, ultimately, reduce the reliability of anomaly detection. To address this issue, we can employ an unfolding technique commonly used in PCA applications [46]. This approach transforms a 3D data matrix (batches I × time steps K × variables J) into a 2D matrix ([batches I × time steps K] × variables J), preserving the temporal dependencies within each batch while enabling cross-batch analysis to detect trends or anomalies. By using this method, the model can effectively process both temporal and batch-level patterns, thus enhancing pattern recognition and anomaly detection. As illustrated in Figure 3.3, batch-wise unfolding—where variable differences between consecutive time steps are analyzed—can only be applied to batches of equal length. In contrast,

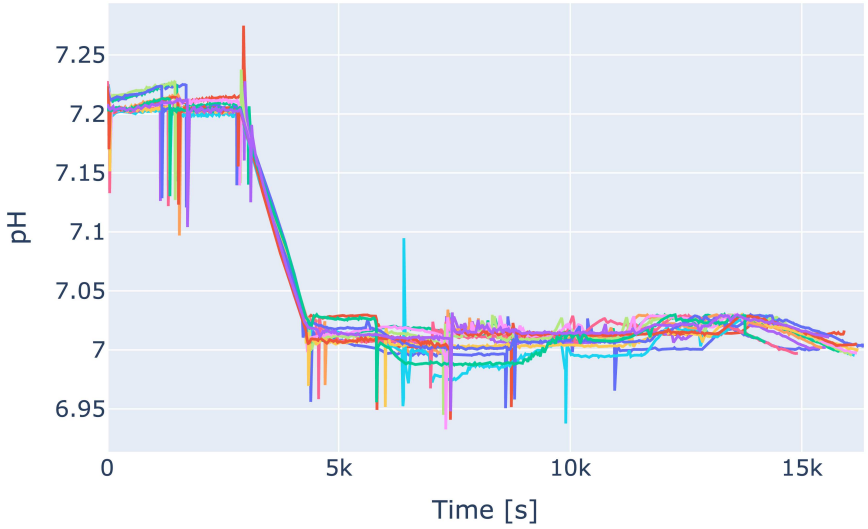


Figure 3.2: Representation of pH sensor readings across different batches, with varying signal durations.

variable-wise unfolding highlights inter-batch patterns, helping to preserve and reveal deviations across batches.

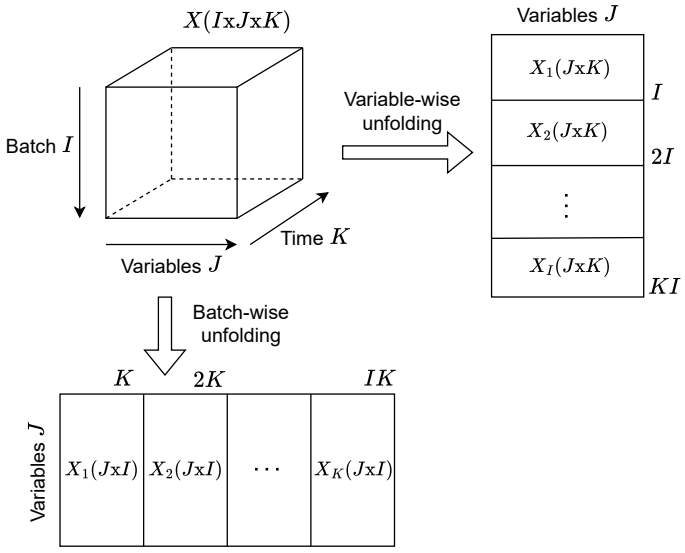


Figure 3.3: Unfolding approaches for batch processes.

3.3.2 Data Preprocessing

To avoid an unbalanced training phase, appropriate data preprocessing is required. In this work, the only required preprocessing step is feature scaling using `StandardScaler`, as the dataset

already consists of pre-selected critical process sensors, ensuring that only the most relevant features are included. `StandardScaler` standardizes the data by subtracting the mean and scaling to unit variance, placing all features on a comparable scale. The transformation is defined as:

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

where x represents the original feature signal, μ is the mean, and σ is the variance. This standardization prevents features with different scales or units from disproportionately influencing the model, thereby improving training convergence and enhancing result interpretability [47].

3.3.3 AutoEncoder (AE)

The AE is an unsupervised, feed-forward neural network that consists of two main components: an encoder and a decoder. The *encoder* compresses high-dimensional input data into a lower-dimensional latent space, effectively extracting the most relevant information. The *decoder* then reconstructs the original data from this compact representation, yielding a similar version of the input data. The model is trained to minimize reconstruction error, which serves as a measure of data fidelity. Additionally, the size of the latent space plays a crucial role in the performance of AEs, as more aggressive compression can lead to a greater loss of information.

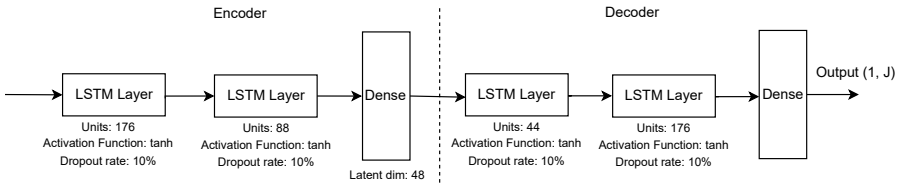


Figure 3.4: Autoencoder structure.

During the anomaly detection phase, the AE evaluates both actual and predicted data points. It continuously monitors reconstruction errors: deviations that exceed an established threshold are flagged as potential anomalies. Figure 3.4 illustrates the proposed model implementation, with the hidden layers and activation functions defined following the optimization phase (detailed in Section 3.3.5). Figure 3.5 demonstrates that the model accurately reconstructs the original signals with overall minimal reconstruction error. However, the error increases in regions where the model struggles to capture rapid variations—such as infrequent peaks or spikes— or sequence variations that were not common in the training batches.

3.3.4 Long-Short Term Memory (LSTM) Regression Model

As defined in Section 1.5.2, LSTM networks are a specialized type of RNN well-suited for modeling sequence data, including time-series and have become one of the most popular and effective RNN architecture for time-series forecasting [48]. Numerous studies have adopted LSTM-based models to predict time-dependent data, demonstrating robust results across diverse application domains [49].

In our framework, the dataset is also fed into the LSTM model. With its inherent capability to capture long-term dependencies, the model learns a mapping from sequences of past observations (inputs) to a continuous target variable (output). Traditional ML models often do not support multi-output prediction, since they are optimized and designed for one-target case. Input and output lags can be particularly beneficial for batch processes, where understanding both recent changes and long-term trends is critical for monitoring and control.

3.3.5 Parameter Optimization

The performance of NN models hinges on their architecture and hyperparameters. To find the best optimal set of parameters, HyperParameter Optimization (HPO) is essential. Bayesan

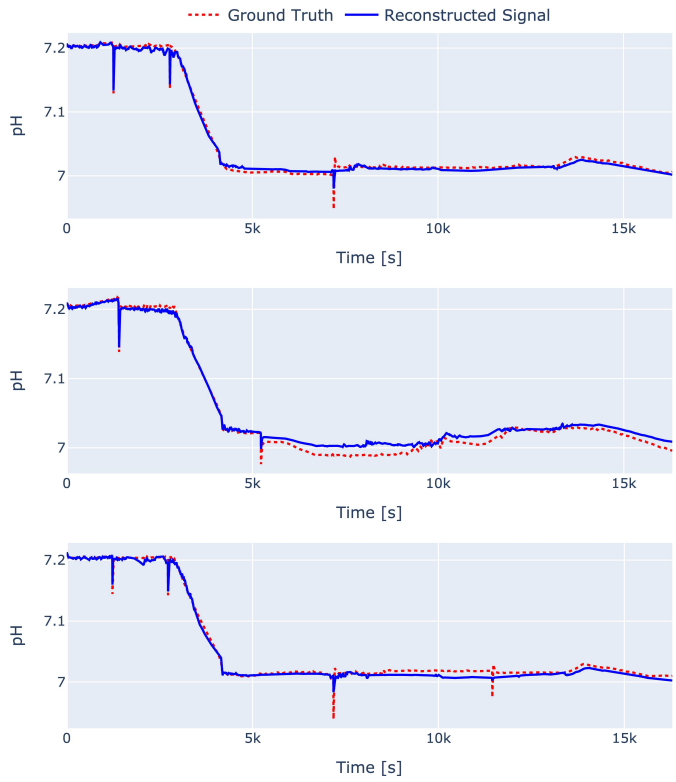


Figure 3.5: Comparison of AE's reconstruction ability of different pH signals.

Optimization (BO) is a Sequential Model-Based Optimization (SMBO) technique well-suited for tuning expensive black-box functions, such as those encountered in DL. In BO, a surrogate model—often a Gaussian Process (GP)—is used to approximate the true objective function. Since evaluating the objective function (for instance, training an ML model) is resource-intensive, the surrogate model significantly reduces computational costs by providing an efficient estimation. BO excels in exploring complex, high-dimensional parameter spaces by effectively balancing exploration and exploitation [50]. These benefits are particularly evident when compared to traditional methods like grid search and random search. Grid search becomes computationally prohibitive as the number of hyperparameters increases due to exponential growth in evaluations, while random search can inefficiently allocate resources by sampling suboptimal regions without focusing on promising configurations. The reconstruction capability of the AE and the prediction accuracy of the LSTM model are optimized using BO, using GP as the surrogate model. The objective is to minimize the Mean Squared Error (MSE) between the input data and the target output—the AE strives to reproduce the input data accurately and the LSTM aims at forecasting future data points effectively. Formally, the optimization problem is defined as follows:

$$\theta^{\star} = \arg\min_{\theta} f(\theta) \tag{3.2}$$

$$f(\theta) = \text{MSE} = \frac{1}{N}\sum_{i=1}^{N} \|x_i - \hat{x}_i\|^2 \tag{3.3}$$

where: $f(\theta)$ is the objective function to be minimized; θ is the hyperparameter configuration vector; x_i and $\hat{x}_i$ represent the ground truth and reconstructed values for each sample i; and N is the total number of samples (with each sample corresponding to a set of sensor readings). The optimal

configuration $\theta^\star$ is the one that yields the lowest observed objective value. To approximate the objective function, the following GP surrogate model is employed:

$$\hat{f}_n(\theta) \sim \mathcal{GP}\big(m(\theta),\, k(\theta,\theta')\big) \tag{3.4}$$

where: n is the current optimization; $m(\theta) = 0$ (normalized data) is the mean function; and the covariance function is defined as:

$$k(\theta,\theta') = \sigma^2 \left(1 + \frac{\sqrt{5}\,r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}\,r}{\ell}\right) \tag{3.5}$$

where: $r = \|\theta - \theta'\|$; $v > 0$ is the smoothness parameter (normally $v = 2.5$); ℓ is the length-scale parameter, which scales the Euclidean distance r; and σ is the variance. Once the surrogate model provides an estimate, a new set of hyperparameters is chosen to further minimize the objective function. This selection is guided by an acquisition function, typically the Expected Improvement (EI) function, defined as:

$$\mathrm{EI}(\theta) = \mathbb{E}\left[\max\left(0, f_{\text{best}} - \hat{f}_n(\theta)\right)\right] \tag{3.6}$$

where $f_{\text{best}} \triangleq \min_{i=1,\ldots,n} f(\theta_i)$ is the best observed value of the objective function up to the current iteration. In this study, the following hyperparameters were optimized, with the MSE as the loss function.

- **Model depth**: determines the number of layers of the model.
- **Units per layer**: specifies the number of neurons in each layer.
- **Activation function**: defines the transformation applied at each layer.
- **Learning rate**: sets the step size for the optimization algorithm.
- **Latent Dimension**: specifies the number of neurons in the encoder's final layer, defining the size of the latent space into which the input is mapped.

The optimal hyperparameter configuration for the training dataset is presented in Figure 3.4, with learning rate equal to $8.6e^{-4}$. This configuration enables the AE to achieve efficient performance as described in Section 3.4.2.

3.3.6 Threshold Computation

In unsupervised anomaly detection models (e.g., AEs), selecting an appropriate threshold for the reconstruction error is critical to distinguish normal variations from true anomalies. One straightforward strategy is to define the anomaly cutoff at a high percentile of the reconstruction error distribution obtained from training data. This approach ensures that only a small fraction (e.g., 1%) of normal data would be mistakenly classified as anomalies by design. This should prioritize catching extreme outliers while limiting false alarms [51]. For multivariate anomaly scoring (or when considering the vector of reconstruction errors across multiple features), thresholding can be based on the Mahalanobis Distance (MD) [52]. The MD measures how far a point is from the center of a distribution while accounting for the covariance structure of the data. The Mahalanobis thresholding approach has the advantage of capturing correlations among variables (or error components), making it more sensitive to unusual combinations of feature values that univariate methods might miss. However, it assumes a reasonably well-estimated covariance matrix; in high dimensions or with limited data, robust covariance estimation or dimensionality reduction may be necessary to apply this method effectively.

In this study, we employ an adaptive threshold technique based on rolling median and MAD of the reconstruction error. The rolling median serves as a local baseline of "normal" behavior, while the rolling MAD provides a scale of typical variability in that period. We then flag a data point as anomalous if its reconstruction error deviates from the current median by more than a chosen factor times the MAD. In other words, the threshold at time t is defined as:

$$\begin{aligned} M_t &= \text{median}\{x_j \mid j \in W(t)\} \\ \text{MAD}_t &= \text{median}\{|x_j - M_t| \mid j \in W(t)\} \\ \tau_t &= M_t + k \times \text{MAD}_t, \end{aligned} \tag{3.7}$$

where: $W(t)$ denotes the set of indices within the rolling window at time t; and k is a scaling factor that adjusts the threshold to suit the dataset's specific requirements. This rolling median/MAD approach yields a time-varying threshold that can adapt to gradual shifts or trends in the data while still being resistant to short-term spikes. It provides a simple, non-parametric way to detect deviations that are extremes with respect to local normal variations by carefully setting the scaling factor and the window size based on data distribution. For our dataset: k is set to 10 to catch anomalies that are not mistaken for new normal observations, caused by frequent spikes in the training data, which may lead to increased reconstruction error; and the length of W equal to 30 to account for local variations.

3.3.7 Evaluation Metrics

To evaluate the performance of the proposed model, we utilize the following two metrics.

- **Mean Squared Error (MSE)**: mainly used in the optimization and training phase, it measures the average squared difference between the predicted values ($\hat{y}_i$) and the true values (y_i). It can be expressed as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.8}$$

  where N is the number of samples.

- **Area Under the Curve Receiver Operating Characteristic (AUCROC)**: it represents the model's ability to distinguish between positive and negative classes. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various thresholds. The AUC is calculated as:

$$\text{AUC} = \int_0^1 \text{TPR}\, d(\text{FPR}) \tag{3.9}$$

  where:

$$\begin{aligned} \text{TPR} &= \frac{\text{TP}}{\text{TP}+\text{FN}} \quad \text{(True Positive Rate)} \\ \text{FPR} &= \frac{\text{FP}}{\text{FP}+\text{TN}} \quad \text{(False Positive Rate)} \end{aligned} \tag{3.10}$$

  The AUCROC is a single scalar value ranging from 0 to 1, where 1 indicates a perfect distinction between normal and anomaly points.

These metrics, together, provide a comprehensive evaluation of the model's accuracy and its ability to explain the variance in the data, as shown in Section 3.4.1.

3.4 Results

In order to provide an overall performance comparison between our proposed AE and both traditional and recent approaches, we leverage ADBench [53], a comprehensive public benchmark for anomaly detection. ADBench evaluates the performance of 30 anomaly detection algorithms, of which 14 are unsupervised, across 57 datasets, encompassing a wide variety of real-world and synthetic scenarios. In this benchmark, anomalies are simulated via four distinct mechanisms: (I) local anomalies, which deviate from the patterns of their immediate neighborhoods; (II) global anomalies, generated by sampling from a uniform distribution; (III) dependency anomalies, where the natural correlations among input features are deliberately disrupted; and (IV) clustered

anomalies, in which anomalous points occur in concentrated groups. This setup allows for a thorough evaluation of model performance under diverse conditions, offering valuable insights of anomaly detection capabilities.

3.4.1 Anomaly Detection with Generic Dataset

As illustrated in Table 3.1, the average AUCROC scores for various unsupervised models—evaluated on datasets characterized by distinct anomaly types—demonstrate that our approach consistently delivers robust results even without specialized tuning. Moreover, further performance improvements are expected following HPO. For instance, while models like k-Nearest Neighbors (k-NN) excel at detecting independent anomalies and PCA proves effective for clustered anomalies, both may struggle when confronted with complex, interrelated anomaly patterns.

Model	Type of Anomaly				
	Original	Local	Global	Dependency	Cluster
IF	**0.7349**	0.8859	0.9973	0.7820	0.9680
One-class SVM	0.6922	0.8618	0.9871	0.6238	0.9569
CBLOF	0.7396	0.8918	0.9970	0.8357	0.8769
COF	0.6437	0.9065	0.9493	0.8877	0.5211
COPOD	0.7177	0.8557	0.9907	0.6065	0.9681
ECOD	0.7187	0.8785	0.9908	0.5950	0.9447
HBOS	0.7122	0.8527	0.9931	0.5982	0.9668
k-NN	0.7058	0.9117	**0.9991**	0.8959	0.8339
LOF	0.6384	**0.9359**	0.9189	0.8814	0.4446
PCA	0.7194	0.8662	0.9933	0.6288	**0.9782**
SOD	0.6880	0.8703	0.9889	0.8940	0.7526
DeepSVDD	0.5612	0.6142	0.7507	0.6133	0.5652
DAGMM	0.6277	0.7927	0.9164	0.6526	0.9354
Proposed AE	0.7242	0.8968	0.9867	**0.8984**	0.8586

Table 3.1: Average AUCROC evaluated on 57 datasets across five anomaly types: Default, Local, Global, Dependency, and Cluster. For each type of anomaly, the best value is highlighted.

3.4.2 Anomaly Detection with GSK Dataset

To assess ADBench's models using our dataset, synthetic anomalies must be incorporated into the test set since the original data contains no outliers and cannot be directly imported into the benchmark without anomalies. Specifically, for each synthetic anomaly, we randomly select one feature and one time step, and inject anomalies corresponding to 1% of the test data size. At the chosen time step, a spike is introduced—its magnitude is determined by the data's standard deviation and scaled by a predefined deterioration factor. Table 3.2 shows AUCROC score of each model with the custom dataset.

3.4.3 Batch Prediction & Reconstruction

Based on the data flow depicted in Figure 3.1, data points are analyzed in two distinct approaches: (I) prediction followed by detection, and (II) real-time detection. For prediction, the window corresponding to each input lag is fed into the LSTM model, which then produces the subsequent outputs, defined by the output lag. To analyze various combinations of input and output values, a grid search is performed over a range of potential input-output lag pairs, depending on the sampling time of 5 minutes. For each pair, a BO optimization is carried out to facilitate a comparative evaluation of the results. By looking at Table 3.3, it is evident that the error increases as the output lag grows with a fixed input lag, caused by the increasing number of predictions and inherently difficulty of multi-point prediction. Although extending the input history does not linearly enhance the model's learning capacity, our findings indicate that a 30-minute input

Method	AUCROC
IF	0.7555
One-class SVM	0.6957
CBLOF	0.9107
COF	0.9581
COPOD	0.6721
ECOD	0.6985
HBOS	0.6643
k-NN	0.9766
LOF	0.9772
PCA	0.6702
SOD	0.8932
DeepSVDD	0.6297
DAGMM	0.6570
Proposed AE	**0.9844**

Table 3.2: AUCROC score with GSK batch dataset.

history (six data points) is sufficient to forecast 15 minutes ahead (three data points). In contrast, predicting 1 hour (twelve data points) accurately requires an input history of 1 hour and 30 minutes (eighteen data points). Figure 3.6 demonstrates that accuracy drops as additional outputs are predicted, with a notable drop in performance after 30 minutes, after which the predictions tend to become stationary. Moreover, Figure 3.9 shows the reconstructed pH signal from a 5-minute prediction step along with its threshold evaluation. Notably, the average AUCROC across the test set with synthetic anomalies is 0.8040, 0.5541, and 0.5507 for the rolling threshold, percentile, and MD methods, respectively.

		Output Predictions			
		1	3	6	12
Input Lag	6	0.0036	**0.0061**	0.0099	0.0209
	12	**0.0033**	0.0067	0.0098	0.0211
	18	**0.0033**	0.0063	0.0094	**0.0191**
	24	**0.0033**	0.0075	**0.0089**	0.0203

Table 3.3: Batch prediction MSE on test data based on input and output lags.

3.5 Conclusions

In this study, we introduced an anomaly detection approach that leverages an LSTM-AE for real-time monitoring in batch processes. The proposed framework addresses the challenges of non-linear dynamics, high dimensionality, and temporal dependencies by using reconstruction error-based detection with a rolling threshold. This method robustly preserves essential information while reducing false positives and detecting gradual deterioration. Validated in real-world scenarios, the LSTM-AE shows promise as an alternative to traditional and ML approaches for identifying subtle, complex anomalies in industrial batch processes. Furthermore, a NN model, based on LSTM layers, is integrated to forecast future data points by analyzing both historical trends and anticipated future steps, providing an accurate prediction of at least 15 minutes ahead. The predictions are then fed into the LSTM-AE, further enhancing its ability to anticipate deviations. This work underscores the potential of DL techniques to revolutionize process monitoring and anomaly detection with advanced predictive capabilities. Future research could explore latent space analysis for improved anomaly prediction, incremental learning with real-time data, and broader deployment across various processes by analyzing critical process variables. Finally, this framework may offer valuable insights in real-time industrial environments into its operational efficiency and scalability, especially when real faults or anomalies are provided to increase model's knowledge, tweaking the threshold accordingly.

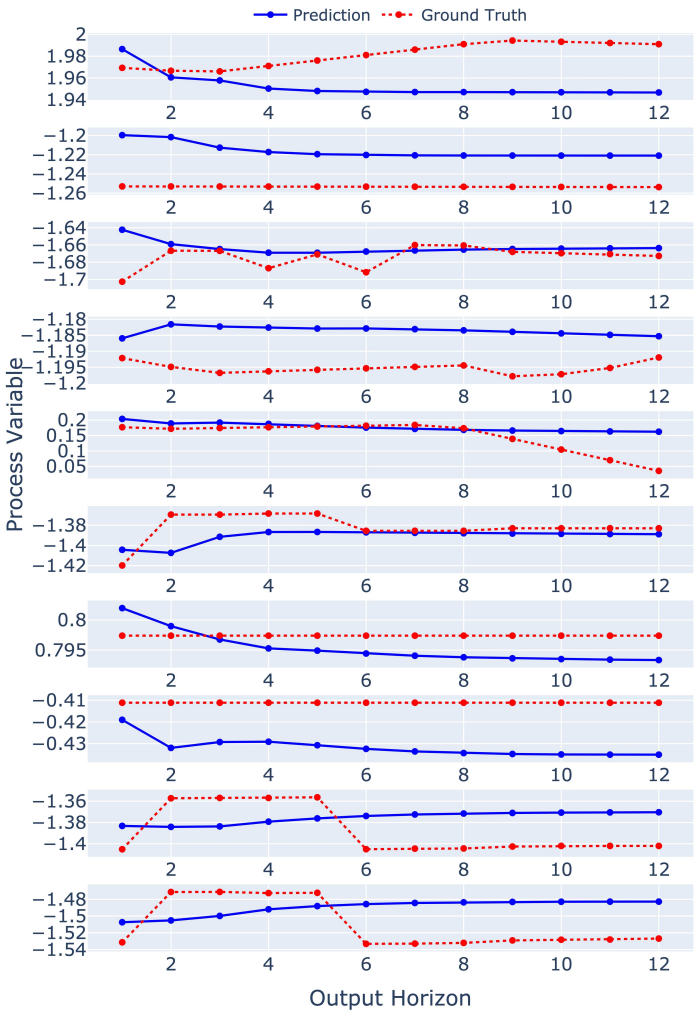


Figure 3.6: Predicted variables across 12 output steps (1 hour) with 1 hour and 30 minutes of input history.

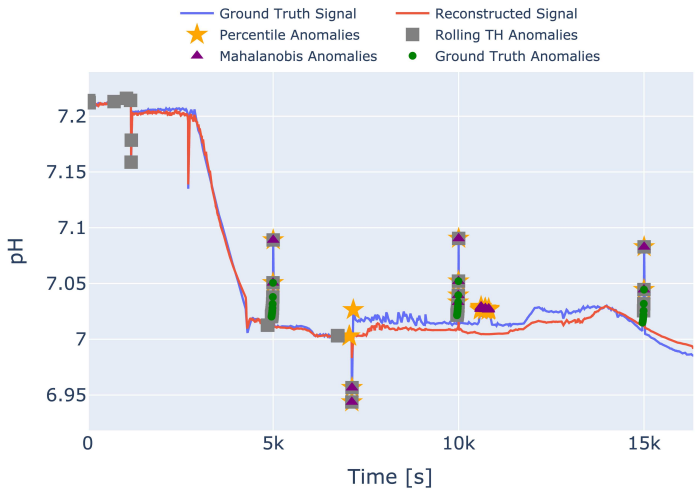


Figure 3.7

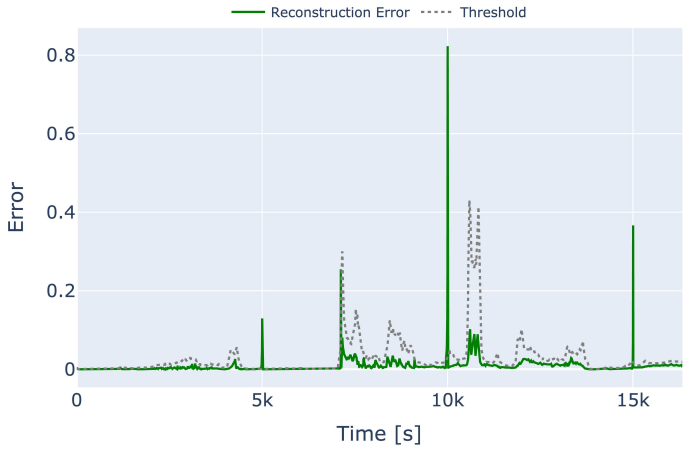


Figure 3.8

Figure 3.9: (a) Comparison of selected threshold mechanism between predicted pH signal and its reconstruction with synthetic anomalies; (b) reconstruction error with the rolling threshold.

Chapter 4

Air Quality Prediction via Embedded ML/DL and Quantized Models

4.1 Introduction

Environmental contamination (and, in particular, air pollution) has been a commonly discussed issue in recent years, with Particulate Matters (PMs)—short particles or droplets suspended in the atmosphere having the potential to be breathed and penetrate the bloodstream—being a major concern due to their harmful impacts on human health [54], [55]. In fact, the World Health Organization (WHO) estimates that illnesses caused by air pollution are responsible for more than 7M premature deaths annually [56]. Furthermore, according to [57], a daily exposure to 10 $\mu g/m^3$ of fine PM increases mortality risk by up to 1% for respiratory diseases, especially with lung cancer (where mortality rate rises by 15 to 27%), and cardiovascular disease mortality by 10%. Additionally, hospital admissions for non-communicable respiratory conditions, namely asthma, pneumonia, flu, and bronchiolitis, increase by up to 4%, 3.9%, and 7%, respectively. air pollution increases the risk of neurodegenerative diseases such as Alzheimer's and Parkinson's [58], [59]. Then, air pollution poses risks also to plants and animals: with the air mainly composed of nitrogen (N_2, about 78%), oxygen (O_2, about 21%, particularly important for plants and animals' growth and development), carbon dioxide (CO_2, in small percentage, useful for the photosynthesis of green plants), water vapor, and other compounds, an excessive increase of pollutants in the air could alter the correct development and lead to pathologies [60]. This further motivates the need to define proactive long-term strategies to mitigate pollution (as much as possible), especially for population well-being [61], [62]. Therefore, such strategies might be enabled through enhanced monitoring systems defined on the basis of Internet of Things (IoT)-oriented well-known architectures and widely adopted processing and communication patterns—e.g., based on Machine Learning (ML), Deep Learning (DL), and quantized models [63]. In particular, this allows to take properly into account non-linear relationships between input variables (i.e., time series [64]) and process large amounts of data, in the end obtaining a higher prediction accuracy than that with traditional approaches [65].

In this research, a comparative performance evaluation (in terms of accuracy-complexity *trade-offs*) of different ML/DL models (namely: Recurrent Neural Network (RNN); Convolutional Neural Network (CNN); Long Short-Term Memory (LSTM); Gated Recurrent Unit (GRU); Legendre Memory Unit (LMU); Temporal Convolutional Network (TCN); Bidirection GRU (BiGRU); Bidirection LSTM (BiLSTM); CNN-GRU, CNN-LSTM, CNN-BiGRU, CNN-BiLSTM) used to predict $PM_{2.5}$-related Air Quality Index (AQI) [66], is discussed. Then, different evaluation metrics (namely: Root Mean Squared Error (RMSE); Mean Absolute Error (MAE); Mean Absolute Percentage Error (MAPE); coefficient of determination (R^2)) and various values of the time lag—corresponding to the number of past observations to be considered to predict the next time step observation value—are considered to maximize estimation accuracy. Finally, a Post-Training Quantization (PTQ) technique is applied to evaluate the feasibility of further optimizing (as much as possible) DL models on *tiny* IoT devices, resource-constrained in terms of memory and computational capabilities.

The main contributions of this research can be summarized as follows: (i) a comparative analysis of different ML and DL models for predicting air quality (in terms of $PM_{2.5}$ concentration) will be presented; (ii) the impact of time lag on the prediction accuracy will be investigated; (iii) the complexity and the memory space required to deploy and run the considered models on *tiny* IoT devices will be considered, in particular evaluating the adoption of PTQ techniques.

The remainder of the research is organized as follows. In Section 4.2, an overview on related works is given. Section 4.4 discusses the proposed data analysis technique. Section 4.5 presents the reference air quality dataset, the experimental performance results, and the PTQ technique.

Finally, in Section 4.6 conclusions are drawn.

4.2 Literature Review

In the following, an overview on air quality prediction techniques proposed in the literature is presented, focusing on ML/DL approaches and on the use of IoT devices, respectively.

4.2.1 Air Quality Prediction through ML and DL Approaches

Several studies focus on RNN architectures [67], such as LSTM [68] and GRU [69], due to their ability to capture temporal dependencies in air quality samples. These models have shown to successfully predict pollutant concentrations with high accuracy.

In [70], a comprehensive comparison between ML algorithms—including Support Vector Machine (SVM), k-Nearest Neighbors (k-NN), and Random Forest (RF)—to predict atmospheric $PM_{2.5}$ levels in the city of Isfahan, Iran, on the basis of data from 7 stations in the time period 2011–2019, with a total amount of 9 features, is presented. According to the obtained results, Artificial Neural Networks (ANNs) hit a 91.1% accuracy, followed by RF, SVM, and k-NN.

Authors in [71] conduct a comparison between LSTM and Seasonal AutoRegressive Integrated Moving Average with eXogenous regressor (SARIMAX) models [72] for $PM_{2.5}$ prediction in Bangkok in the time period 2017–2018. Their results show that LSTM achieves lower (in terms of 24 hour average value) RMSE and MAE values equal to 6.04 units and 4.86 units, respectively, while SARIMAX returns values equal to 7.99 units and 5.74 units, respectively.

In [73], a model, denoted as Aggregated LSTM (ALSTM), allows to forecast the concentration of $PM_{2.5}$ in five distinct industrial air quality sites in Taiwan. In detail, the adopted dataset consists of 17 environmental features collected in the time period 2012–2017, while the proposed model consists of stacked LSTM layers followed by fully-connected dense layers. The ALSTM model demonstrates superior accuracy when compared to LSTM, Gradient Boosted Tree Regression (GBTR) [74], and Support Vector Regression (SVR) [75] models.

Authors in [76] target the prediction of $PM_{2.5}$ in eight Korean cities, taking into account a dimensionality problem caused by a dataset with numerous variables and limited observations, and reducing it using RNN, LSTM, and BiLSTM models with Principal Component Analysis (PCA) [77]. The obtained results show that using PCA with LSTM and BiLSTM reduces RMSE and MAE by 16.6% and 33.3%, respectively.

In [78], a CNN-LSTM model is proposed in order to predict the $PM_{2.5}$ concentration in Beijing considering spatio-temporal correlations, over a dataset composed of 20,757 pollutant concentration and meteorological parameters' observations collected in the time period January 2015–March 2020. According to the obtained results, CNN-LSTM outperforms Multi-Layer Perceptron (MLP) and LSTM. Moreover, a combination of one-dimensional (1D) Convolutional and BiGRU (CBGRU) model for $PM_{2.5}$ short-time prediction is proposed in [79]. In detail, the chosen dataset is composed of 43,800 8-feature hourly observations collected in the time period January 2010–December 2014, while the proposed model is composed of two 1D convolutional layers followed by two BiGRU layers. According to the obtained results, CBGRU provides a lower prediction error in comparison to DL models (including LSTM, GRU, and RNN). Similarly, the performances of LSTM, GRU, and BiLSTM are compared in [80], in order to predict $PM_{2.5}$ in three Korean cities (namely: Seoul, Daejeon, and Busan), using a 1-hour interval dataset collected in the time period May 2014–December 2021. The presented results show that BiLSTM is the best performing in terms of long-term predictions.

In [81], a stacked LSTM model is proposed to predict the next hour $PM_{2.5}$ concentration in the city of Istanbul, Turkey, training the model over a dataset collected in the time period January 2015–November 2019. According to the obtained results, the proposed model outperforms LSTM, RNN, and GRU, with an RMSE equal to 25.20 units.

Finally, a 1D-CNN-BiLSTM model with parallel input from the target and neighboring monitoring stations is presented in [82] for the hourly prediction of $PM_{2.5}$. As a result, the suggested method chooses neighboring stations based on distance and wind statistics, proving to be effective when a combined use of target and chosen monitoring stations is performed.

Study	Adopted technique	HPO	IoT deployability evaluation	Quantization
[70]	ANN	✗	✗	✗
[71]	LSTM	✗	✗	✗
[73]	Three combined LSTM	✗	✗	✗
[76]	PCA-BiLSTM	✗	✗	✗
[78]	CNN-LSTM	✗	✗	✗
[79]	CNN-BiGRU	✗	✗	✗
[81]	Stacked LSTM	✗	✗	✗
[82]	1D-CNN-BiLSTM	✗	✗	✗
[83]	XGBoost	✓	✓	✗
[84]	GRU	✗	✓	✗
Proposed model	CNN-BiGRU	✓	✓	✓

Table 4.1: Literature works categorized along their main features.

4.2.2 Air Quality Prediction on Resource-Constrained Devices

A tiny ML model effective in predicting the AQI using trained Decision Tree (DT), RF, and XGBoost models is proposed in [83]. Although faced with computational limitations, the authors achieve an accuracy of 75.2% adopting XGBoost, that outperforms both RF and DT while complying with a 2 MB memory limitation. Similarly, in [85] a low-cost device—based on a Raspberry Pi Pico as the main micro controller—for air quality monitoring and prediction is proposed. In detail, two models (composed of 2D convolutional layers making an AE) are used: the first model forecasts CO_2, temperature, and humidity, by analyzing six hours of past observations; the second model (namely, a model imputer [86]) is used for missing values imputation. Then, with model predictor's and model imputer's sizes of 107,708 and 126,536 B, respectively, the light models' size decrease by 47.7% and 56.6%, respectively.

Finally, in [84] a comparative analysis and performance evaluation (in terms of computational complexity) of different ML and DL models on a tiny IoT device to forecast the $PM_{2.5}$ concentration is proposed. According to the obtained results, GRU provides the most accurate prediction with respect to the other considered models.

Statement of Contribution

For the sake of clarity, a tabular comparison between the previously detailed literature works and the approach proposed in this research is shown in Table 4.1. In the following, the key contributions of our proposed air quality prediction mechanism are listed.

1. This research provides a comprehensive evaluation of various DL models in terms of both prediction accuracy and computational complexity, addressing *trade-offs* required for their deployment on resource-constrained devices.
2. This research discusses the implementation of 8-bit quantization to evaluate its influence on the prediction performance, providing insights into the effects of model compression on accuracy and efficiency.
3. This research highlights the use of HyperParameter Optimization (HPO) to identify the optimal architecture and hyperparameters, thus improving both performance and resource efficiency.
4. This research focuses on IoT deployability by evaluating models for real-world implementation, demonstrating their practical feasibility in edge computing environments.

4.3 Background

Before investigating the proposed air quality estimation and prediction model (detailed in Section 4.4), as well as the experimental performance (discussed in Section 4.5), in the following

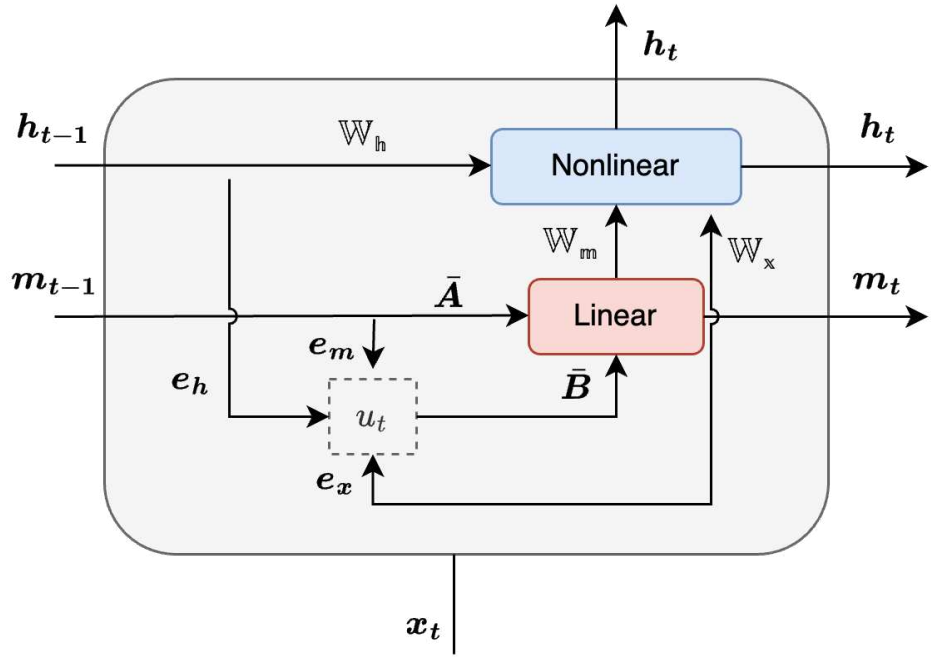


Figure 4.1: Simplified mathematical model of an LMU.

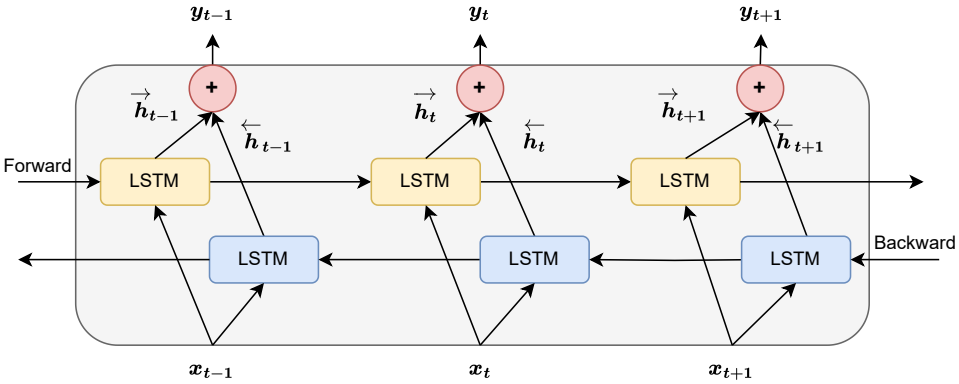


Figure 4.2: Simplified mathematical model of a BiRNN.

an overview on the considered ML and DL models is presented, to highlight their features. In addition to those detailed in Section 1.5 the following are considered for time-series prediction.

4.3.1 Legendre Memory Unit (LMU)

Legendre Memory Unit (LMU) is a recently introduced RNN using less computational resources to preserve long-range time dependencies. In detail, LMU cells break down the time history into d Ordinary Differential Equations (ODEs) exploiting Legendre polynomials and Fourier attributes to orthogonalize the continuous-time history of its encoded input signal (denoted as $u_t \in \mathbb{R}$) [87]. For the sake of completeness, a LMU (whose block architecture is shown in Figure 4.1) can be defined as follows:

$$u_t = e_x^T x_t + e_h^T h_{t-1} + e_m^T m_{t-1} \tag{4.1}$$

$$m_t = \bar{A} m_{t-1} + \bar{B} u_t \tag{4.2}$$

$$h_t = f(\mathbb{W}_x x_t + \mathbb{W}_m m_t + \mathbb{W}_h h_{t-1}) \tag{4.3}$$

where: $f(\cdot)$ is a non-linear function (such as tanh); $\mathbb{W}_x \in \mathbb{R}^{n_h \times n_x}$, $\mathbb{W}_h \in \mathbb{R}^{n_h \times n_h}$, $\mathbb{W}_m \in \mathbb{R}^{n_h \times d}$ represent the weight matrices; $e_x \in \mathbb{R}^{n_x}$, $e_h \in \mathbb{R}^{n_h}$, $e_m \in \mathbb{R}^d$ are the encoding vectors; x_t represents the input at the current time step t; $m_t \in \mathbb{R}^d$ denotes the unit's linear memory; h_t represents the non-linear hidden state; $\bar{A} \in \mathbb{R}^{d \times d}$ and $\bar{B} \in \mathbb{R}^{d \times 1}$ correspond to the discretized matrices provided by ODEs; n_h and n_x are the dimension of the hidden state and input vectors, respectively.

4.3.2 Bidirectional RNN (BiRNN)

In a Bidirectional RNN (BiRNN), data sequences are processed in two directions, namely *forward* and *backward*. This motivates the reason why designing Bidirectional RNNs (BiRNNs) generally comprises two hidden layers at each time step (as shown in Figure 4.2): the *first* layer is used to process the sequence in the forward direction, while the *second* layer is expedient to process the sequence in the backward direction.

4.3.3 Overall Discussion

Overall, the ML/DL models previously discussed and considered in the further experimental performance evaluation are particularly well-suited for correlated sequential data with multiple inputs and outputs. Moreover, TCN and LMU have been included as novel models capable of efficiently capturing long- and short-term temporal dependencies. In fact, traditional ML models (such as AdaBoost, DT and RF) are susceptible to overfitting, particularly when the models are complex. As the model complexity increases, both the training and prediction times tend to increase significantly [88], [89]. Furthermore, they often face challenges in complex scenarios involving long-term information [90]. Then, as will be better highlighted in Subsection 4.5.4, including a quantization layer in the processing pipeline for DL models ensures an optimized deployment for constrained devices.

4.4 Proposed Model

On the basis of DL models detailed in Section 1.5 and Section4.3, in the following the different steps defined in the proposed comparative performance analysis are discussed, namely: data preparation (detailed in Subsection 4.4.1); Bayesian Optimization (BO, detailed in Subsection 4.4.2); and a sliding window technique (detailed in Subsection 4.4.3).

4.4.1 Data Preparation

The data preparation step includes data correlation analysis (Subsection 4.4.1.1) and data normalization (Subsection 4.4.1.2), as follows.

4.4.1.1 Data Correlation Analysis

In order to perform a data correlation analysis, the Pearson Correlation Coefficient (PCC) [91] has been adopted. In detail, PCC (denoted as ρ) is a statistical metric accurately measuring importance and trend of the dependence between two variables, and defined as

$$\rho = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{4.4}$$

where: n corresponds to the number of observations; X_i and Y_i are individual data points; $\bar{X}$ and $\bar{Y}$ represent the average values of the random variables X and Y, respectively. To this end, PCC can range from -1 to 1: two variables with the same direction of variation lead to the highest positive correlation ($\rho = 1$); two variables with opposite directions of variation lead to the highest (in norm) negative correlation ($\rho = -1$); finally, the absence of a linear dependency between the considered variables leads to $\rho = 0$. For the sake of completeness, the PCC obtained on the features still present in the adopted dataset—further described in Subsection 4.5.1—is shown in Figure 4.3.

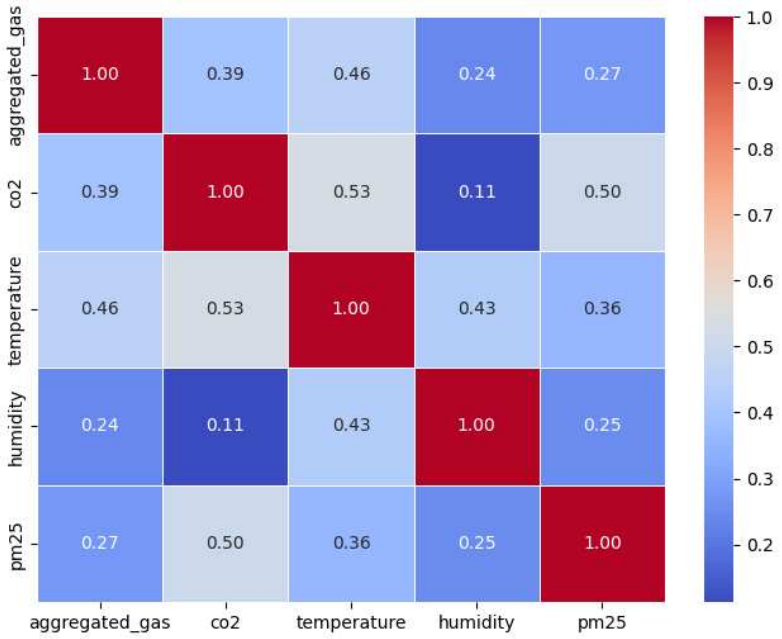


Figure 4.3: PCC of the features of the adopted dataset.

Looking at the PCC values shown in Figure 4.3, it can be observed that each feature exhibits a weak or moderate positive correlation with the others. This indicates that all features contribute complementary information to predict $PM_{2.5}$ levels. Moreover, with this limited information set, a feature augmentation may result in noise, potentially leading to overfitting and increasing model complexity. Consequently, only the declared features are retained for models training.

4.4.1.2 Data Normalization

Data normalization eases the training process of NNs by ensuring that the different features representing the candidate dataset are mapped to a comparable scale, thus accelerating the convergence of different algorithms while enhancing the stability and efficiency of the proposed model. For this reason, the MinMaxScaler function [92], adopted in the proposed performance analysis, allows to map input data within a specific range (typically $[0,1]$ or $[-1,1]$) and can be defined as

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{4.5}$$

where: x and x_{scaled} represent original and scaled values of a specific feature, respectively; $x_{\min}$ and $x_{\max}$ correspond to the minimum and maximum values of that feature, respectively. In fact, the MinMaxScaler function is applied in order to prevent data distortion from changes in standard deviation, as it reshapes the data to fit within a specified range and preserves data sparsity.

4.4.2 Bayesan Optimization (BO)

BO represents a popular method for hyperparameter tweaking in DL models, as it allows performance optimization, reducing overfitting and underfitting, and improving generalization, resource efficiency, and durability [93]. These benefits become more and more evident when compared to traditional techniques (e.g., grid search and random search), which show significant limitations. In particular, grid search becomes computationally inefficient because of an exponential growth of the number of evaluations, as the number of hyperparameters increases, while random search may waste resources by exploring poorly performing regions and failing to focus on optimal configurations [94]. In contrast, BO performs a global optimization of black-box functions by employing a probabilistic model, typically a GP, to more effectively explore the

search space [95]. To this end, the HPO task can be formally defined as:

$$x^{\star} = \arg\min_{x \in X} f(x) \tag{4.6}$$

where $f(x)$ represents the objective function to be minimized, and x denotes the vector of hyperparameter configurations within the search space X. Consequently, the set of hyperparameters $x^{\star}$ will yield the minimal objective function value observed thus far.

For completeness, the following hyperparameters have been optimized: learning rate, units per layer, batch size, and activation function. More in detail, they have been chosen due to their significant impact on models' performance, efficiency, and generalization capabilities. Thus, in order to guide the search, we employed the Expected Improvement (EI) acquisition function—calculating the potential for improvement at each new point in the search space and focusing the optimization on promising areas—defined as follows:

$$\mathrm{EI}(x) = \mathbb{E}\left[\max\left(0, f(x^{\text{best}}) - f(x)\right)\right] \tag{4.7}$$

where: x^{best} corresponds to the set of hyperparameters providing the best objective function up to the current optimization step. To this end, the hyperparameter search space was carefully defined to ensure a comprehensive exploration. With regard to the learning rate, we set a continuous range from 10^{-5} to 10^{-3}, while the number of units per layer was optimized with integers ranging from 32 to 128, the batch size was optimized with integers ranging from 16 to 32, and the activation function was chosen between ReLU and tanh.

In the following a performance comparison of the BO technique using different surrogate models, namely GPs and RFs [96], [97], is detailed. Then, since in BO a surrogate model offers an approximation of the actual objective function under optimization, correlating input data to output data, when the actual relationship is either unknown or computationally costly, in order to compare original and surrogate models, the accuracy is evaluated through the RMSE, chosen as optimization metric and described in Subsection 4.5.2. In particular:

- $\mathrm{RMSE}_{\mathrm{GP}}$ represents the RMSE obtained using a GP surrogate model;
- $\mathrm{RMSE}_{\mathrm{RF}}$ represents the RMSE obtained using a RF surrogate model.

Hence, a ratio (denoted as r) between GPs and RFs, with BO serving as the baseline model, is calculated as follows:

$$r = 1 - \left(\frac{\mathrm{RMSE}_{\mathrm{GP}}}{\mathrm{RMSE}_{\mathrm{RF}}}\right) \tag{4.8}$$

where: $r > 0$ indicates that GPs outperform RFs ($\mathrm{RMSE}_{\mathrm{GP}} < \mathrm{RMSE}_{\mathrm{RF}}$); while $r < 0$ shows an improved performance of RFs compared to GPs ($\mathrm{RMSE}_{\mathrm{GP}} > \mathrm{RMSE}_{\mathrm{RF}}$).

In Table 4.2, the ratio r obtained by all the considered ML/DL models is shown, considering various values of $\mathcal{W}$, which corresponds to the time lag. From the obtained results, it can be observed how GPs yield an average improvement (in terms of model accuracy) equal to 0.37%, at the same time not increasing the model's complexity. GP surrogates are particularly effective where the hyperparameter space is low-dimensional and composed mainly of continuous parameters [98]. In contrast, RF surrogates naturally manage high-dimensional spaces that include a mix of continuous, discrete, categorical, and conditional variables. The inherent structure of ensemble trees allows RF models to scale better as the hyperparameter space increases while also providing a lower computational cost [99].

Ultimately, the decision to use GP or RF surrogate models in BO depends primarily on the characteristics of the hyperparameter space. In our study, by focusing on optimizing four hyperparameters, GP models can explore more efficiently and at a lower computational cost, thereby narrowing the gap between GP and RF approaches.

4.4.3 Sliding Window Technique

Finally, a sliding window technique has been adopted in the proposed model to convert time series data into sequential data, in order to forecast $PM_{2.5}$ levels for both short-term and long-term time

Model	RMSE Ratio r [%]			
	$\mathcal{W} = 5$	$\mathcal{W} = 10$	$\mathcal{W} = 15$	$\mathcal{W} = 20$
BiGRU	0,43%	0,07%	1,05%	-1,18%
BiLSTM	-1,93%	-0,47%	3,58%	-0,32%
CNN	1,85%	2,37%	1,20%	-0,09%
CNN-BiGRU	-0,91%	2,06%	1,13%	-7,35%
CNN-BiLSTM	-1,89%	-1,96%	-0,57%	-17,28%
CNN-GRU	-0,07%	0,22%	-0,17%	-1,09%
CNN-LSTM	3,29%	5,77%	2,31%	8,49%
GRU	-1,29%	-1,44%	0,49%	1,20%
LSTM	-1,62%	6,95%	3,23%	5,84%
RNN	3,45%	0,83%	0,58%	-0,21%

Table 4.2: Accuracy of the considered ML and DL models in terms of the ratio r between GPs and RF considering different time lags $\mathcal{W}$.

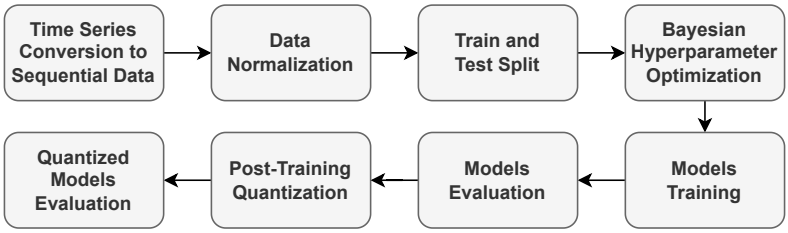


Figure 4.4: Flowchart of the proposed prediction process.

horizons. In particular, this approach involves the evaluation of several time lags $\mathcal{W}$ to identify the setting which enables the model to adequately represent temporal dependencies. The selected time lag allows to identify patterns and relationships between past $PM_{2.5}$ concentrations and future values by examining historical data through the sliding window. By integrating the relevant information from several time intervals, the sliding window approach improves the model's precision in predicting $PM_{2.5}$ levels across different time horizons.

For the sake of completeness and clarification, a graphical representation of the proposed prediction process (returning the experimental results further detailed in Section 4.5) is shown in Figure 4.4, while the detailed architectures of the proposed models are listed, considering various values of the time lag $\mathcal{W}$, in Table 4.3.

4.5 Results

4.5.1 Dataset

The experimental dataset (publicly available at [100]) is composed of air quality measurements collected in an interior travelers transit area at the Brindisi airport, in the south of Italy [101] in the time period October 2022–October 2023. In detail, three inexpensive commercial sensors—namely, Adafruit MiCS5524, Sensirion SCD30, and Sensirion SPS30—have been used to collect air quality data, such as CO_2, temperature, humidity, aggregated gas levels, and $PM_{2.5}$, with a 2 sec sampling period. By employing the sliding window mechanism (detailed in Subsection 4.4.3) there has been the flexibility to re-sample the dataset at different sampling periods. The considered combinations of values of the sampling period (denoted as t_{sampling}, dimension: [s]), the prediction horizon (denoted as s, dimension: [h]), and the number of predictions (denoted as n_{pred}, adimensional) are listed in Table 4.4. Moreover, we remark that an overlapping time lag mechanism was used during the preparation of both training and test sets: this increases the number of input samples available in the experimental evaluation stage.

In the following, the performance with the dataset re-sampled as a 1-hour dataset (t_{sampling} = 1 hour) are investigated. The chosen combinations guarantee a sufficient level of

Model	$\mathcal{W} = 5$	$\mathcal{W} = 10$	$\mathcal{W} = 15$	$\mathcal{W} = 20$
BiGRU	neurons = 113 activation = relu	neurons = 128 activation = tanh	neurons = 32 activation = relu	neurons = 128 activation = relu
BiLSTM	neurons = 92 activation = tanh	neurons = 128 activation = relu	neurons = 32 activation = relu	neurons = 128 activation = relu
CNN	filters = 118 kernel_size = 3 activation = relu	filters = 128 kernel_size = 3 activation = relu	filters = 128 kernel_size = 3 activation = relu	filters = 128 kernel_size = 3 activation = tanh
CNN-BiGRU	neurons/filters = 68 kernel_size = 3 activation = relu	neurons/filters = 46 kernel_size = 3 activation = relu	neurons/filters = 128 kernel_size = 3 activation = relu	neurons/filters = 32 kernel_size = 3 activation = relu
CNN-BiLSTM	neurons/filters = 73 kernel_size = 3 activation = relu	neurons/filters = 128 kernel_size = 3 activation = tanh	neurons/filters = 128 kernel_size = 3 activation = tanh	neurons/filters = 128 kernel_size = 3 activation = relu
CNN-GRU	neurons/filters = 114 kernel_size = 3 activation = relu	neurons/filters = 83 kernel_size = 3 activation = relu	neurons/filters = 100 kernel_size = 3 activation = relu	neurons/filters = 128 kernel_size = 3 activation = relu
CNN-LSTM	neurons/filters = 98 kernel_size = 3 activation = relu	neurons/filters = 66 kernel_size = 3 activation = tanh	neurons/filters = 110 kernel_size = 3 activation = tanh	neurons/filters = 116 kernel_size = 3 activation = tanh
GRU	neurons = 32 activation = relu	neurons = 128 activation = relu	neurons = 84 activation = relu	neurons = 89 activation = relu
LMU	neurons = 96 activation = relu	neurons = 64 activation = relu	neurons = 128 activation = tanh	neurons = 128 activation = tanh
LSTM	neurons = 94 activation = relu	neurons = 128 activation = tanh	neurons = 128 activation = tanh	neurons = 56 activation = relu
RNN	neurons = 110 activation = relu	neurons = 128 activation = relu	neurons = 95 activation = tanh	neurons = 126 activation = relu
TCN	neurons/filters = 70 kernel_size = 3 activation = relu	neurons/filters = 119 kernel_size = 3 activation = tanh	neurons/filters = 128 kernel_size = 3 activation = relu	neurons/filters = 115 kernel_size = 3 activation = relu

Table 4.3: Bayesian hyperparameters for the different time lags $\mathcal{W}$.

t_{sampling}	Prediction horizon s [hr]	n_{pred} [num]
1 hr	8	8
30 min	3	6
2 min	1	30

Table 4.4: Combinations of sampling time t_{sampling}, prediction horizon s, and number of predictions n_{pred}, applied on the chosen air quality dataset.

prediction granularity. As expected, re-sampling the dataset with t_{sampling} = 2 min allows to predict $PM_{2.5}$ variations in the subsequent hour with a finer detail than in the case with t_{sampling} = 1 hour. Considering t_{sampling} = 30 min allows to improve the trend estimation accuracy. Nevertheless, even if values lower than 1 hour (e.g., 2 min or 30 min) can be employed for trend analysis, they are not suitable for long-term prediction. On the other end, considering values of t_{sampling} longer than 1 hour will degrade the performance, leading to a smaller dataset.

4.5.2 Evaluation Metrics

The performance of the considered algorithms (detailed in Section 1.5) is evaluated on the basis of RMSE, MAE, MAPE, and R^2, defined as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.9}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.10}$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{4.11}$$

$$\tag{4.12}$$

where: n corresponds to the number of data points in the test set; y_i represents the actual value; $\hat{y}_i$ represents the corresponding predicted value; and $\bar{y} = 1/n\sum_{i=1}^{n} y_i$ is the mean value of data points.

Overall, the chosen evaluation metrics highlight different behaviors of the considered models. More precisely, two models may have nearly identical MAE/RMSE, yet greatly differ in terms of R^2, thus indicating distinct residual-variance patterns. Conversely, a model having a higher R^2 but a larger MAE can return lower performance because of short-term volatility. Together, RMSE and MAE are sensitive to outliers, while MAPE adds a scale-free, relative-error perspective. More in detail, RMSE and MAE are measured on the basis of the predicted feature, i.e., $PM_{2.5}$, while MAPE is expressed as a percentage of the error of $PM_{2.5}$, and R^2 is adimensional.

As shown in Figure 4.5, RMSE has been designated as reference metric for both training and testing phases. Then, each metric is assessed for various values of the time lag $\mathcal{W}$, leading to a fair performance comparison of the considered ML and DL models in the test set. In fact, less architectural complex models may experience good overall performance with small values of $\mathcal{W}$, but may face challenges in identifying correlations for long-term predictions, particularly with larger input dimensions, compensating with higher complexity layers. In most cases, increasing the historical knowledge leads to improved accuracy across various models, as shown in Figure 4.6, with CNN-BiGRU returning the best accuracy among the considered models.

It might also be reasonable for model predictions to become less accurate for increasing values of the prediction horizon s: if s is extended, then the model needs to identify additional correlations within the data and this may become increasingly challenging and computationally heavy. As a reference, in Figure 4.7 all the metrics for a given dataset and $\mathcal{W} = 20$ are shown, while Figure 4.8 shows the increasing prediction error as a function of the time horizon. It can be observed that, by increasing s, ML and DL models lose their ability to predict variations, while they still capture the trend in long-term predictions.

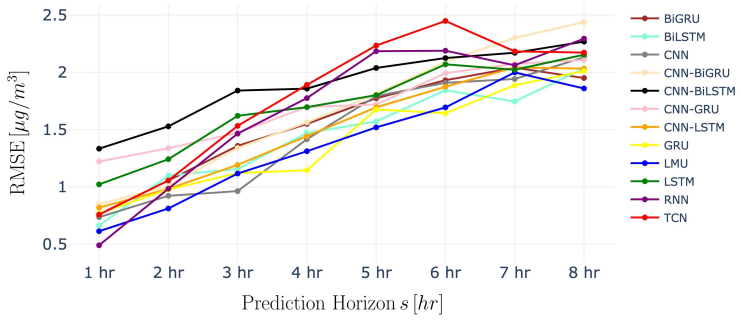
Finally, Figure 4.9 shows the performance of various models across three key metrics—namely, average MAE, MAPE, and RMSE—evaluated over all the considered time lags $\mathcal{W} = \{5, 10, 15, 20\}$ and all the considered prediction horizons $s \in \{1, \ldots, 8\}$. The obtained results demonstrate that CNN-BiGRU achieves, on average, the lowest error rate (in terms of MAE, MAPE, and RMSE), thus reducing the prediction error compared to other DL models. This superior performance highlights its effectiveness in capturing both spatial and temporal dependencies for $PM_{2.5}$.

4.5.3 Model Complexity

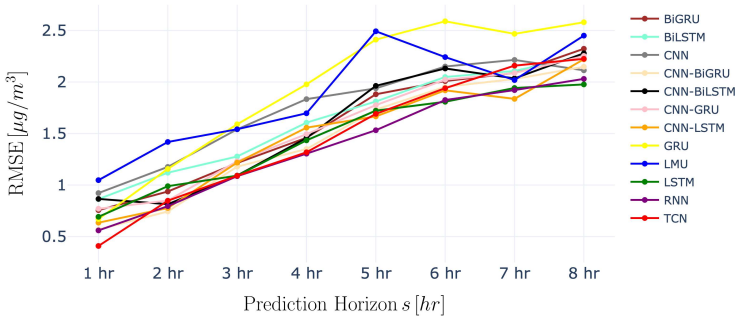
In order to better focus on the utilisation of ML and DL models by IoT nodes (often characterized by limited, if not constrained, resources), in addition to the performance analyzed in Subsection 4.5.2, the computational complexity of the considered models has been further investigated considering the number of Multiply and ACCumulate (MACC) operations (denoted as n_{MACC}, adimensional), the RAM usage (dimension: [KiB]), and the execution time (denoted as t_{exec}, dimension: [ms]), defined as follows:

$$t_{\text{exec}} = \frac{n_{\text{MACC}} \cdot 11}{f_{\text{board}}} \tag{4.13}$$

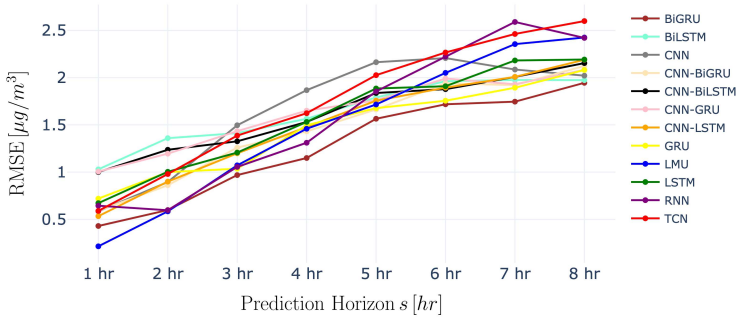
where: f_{board} corresponds the frequency of the STM32F469I-DISCO board [102], equal to 180 MHz; n_{MACC} is multiplied by 11 because of the requirement of 11 cycles per MACC (on average, with minimum and maximum values equal to 8.688 and 16.487, respectively, as further detailed in Table 4.5 and shown in Figure 4.10) on that board—the value of 11 has been achieved considering the average n_{MACC} for all of the evaluated models, except for LMU and TCN.



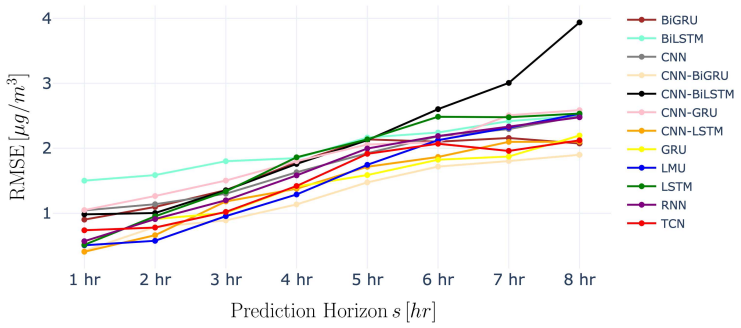
(a)



(b)

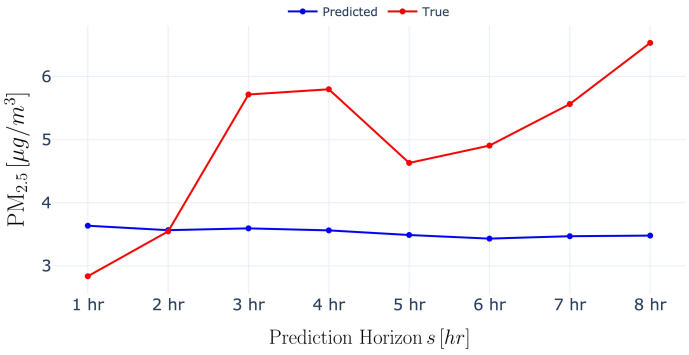


(c)

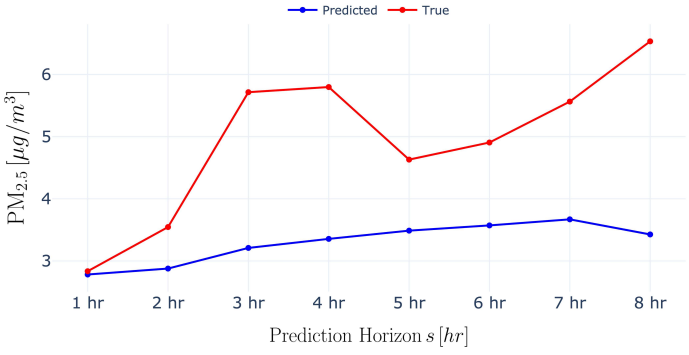


(d)

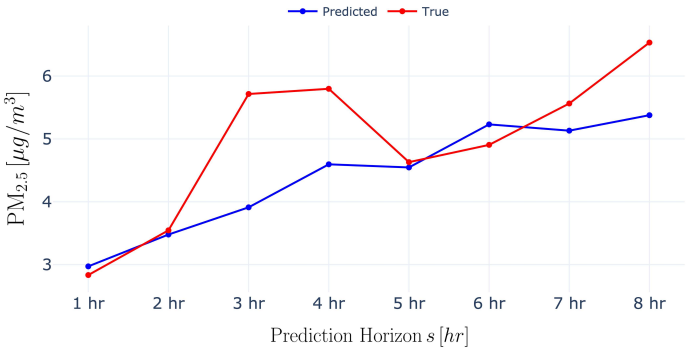
Figure 4.5: Performance of the considered ML and DL models on the basis of RMSE (with lower values of RMSE being the better) for different values of the time lag $\mathcal{W}$ in the test set: (a) $\mathcal{W} = 5$, (b) $\mathcal{W} = 10$, (c) $\mathcal{W} = 15$, (d) $\mathcal{W} = 20$.



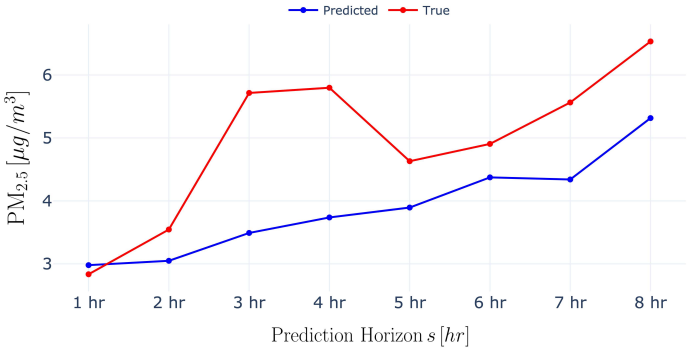
(a)



(b)



(c)



(d)

Figure 4.6: Performance of the CNN-BiGRU model for different values of the time lag $\mathcal{W}$: (a) $\mathcal{W} = 5$, (b) $\mathcal{W} = 10$, (c) $\mathcal{W} = 15$, (d) $\mathcal{W} = 20$.

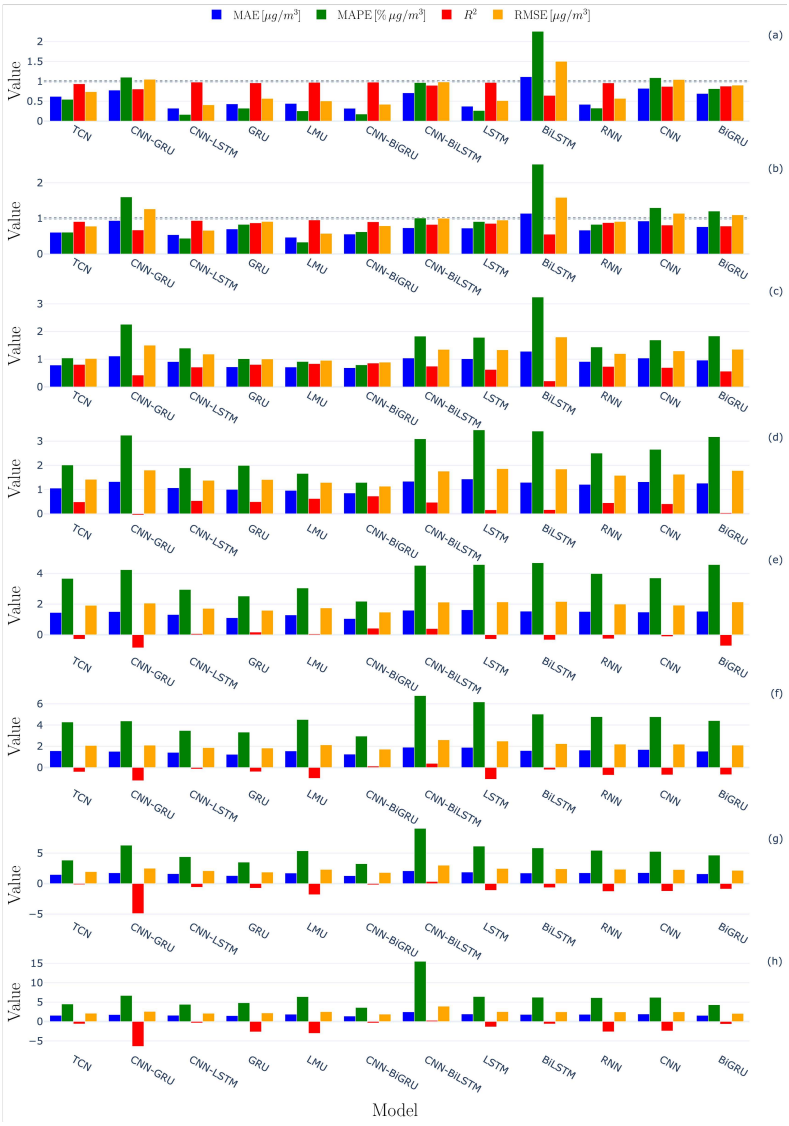


Figure 4.7: Evaluated metrics for each analyzed ML and DL model considering a time lag $\mathcal{W} = 20$ across each prediction hour: (a) 1-hour ($s = 1$), (b) 2-hour ($s = 2$), (c) 3-hour ($s = 3$), (d) 4-hour ($s = 4$), (e) 5-hour ($s = 5$), (f) 6-hour ($s = 6$), (g) 7-hour ($s = 7$), (h) 8-hour ($s = 8$).

Model	$\mathcal{W} = 5$	$\mathcal{W} = 10$	$\mathcal{W} = 15$	$\mathcal{W} = 20$
BiGRU	10.045	10.261	14.571	9.924
BiLSTM	12.065	10.462	16.487	10.381
CNN	10.426	11.779	11.073	11.985
CNN-BiGRU	10.475	11.608	9.197	12.416
CNN-BiLSTM	10.630	9.847	9.513	9.515
CNN-GRU	9.421	10.032	9.962	9.354
CNN-LSTM	9.963	11.619	10.310	10.272
GRU	14.231	9.879	10.820	11.181
LSTM	11.145	11.013	11.062	14.723
RNN	8.836	8.688	10.608	8.763

Table 4.5: Evaluation cycle per MACC for various values of the time lags $\mathcal{W}$.

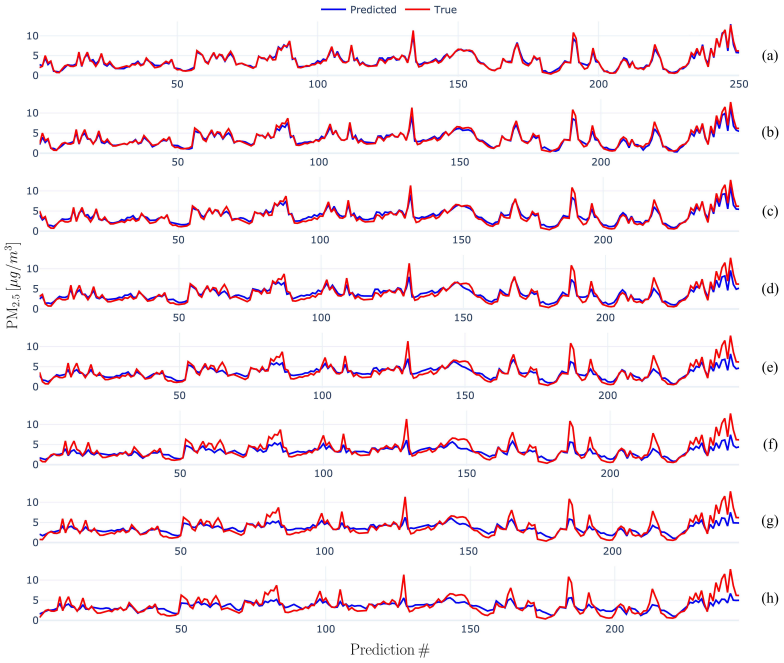


Figure 4.8: Prediction performance of the CNN-BiGRU model with a time lag $\mathcal{W} = 20$ across each prediction hour: (a) 1-hour ($s = 1$), (b) 2-hour ($s = 2$), (c) 3-hour ($s = 3$), (d) 4-hour ($s = 4$), (e) 5-hour ($s = 5$), (f) 6-hour ($s = 6$), (g) 7-hour ($s = 7$), (h) 8-hour ($s = 8$).

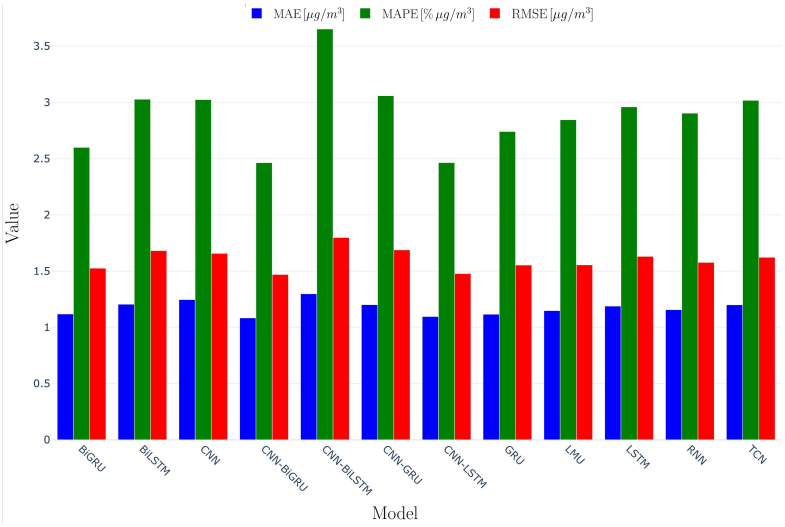


Figure 4.9: Average MAE, MAPE, and RMSE of the evaluated DL models considering all the time lags $\mathcal{W} = \{5, 10, 15, 20\}$ and all the prediction horizons $s \in \{1, \ldots, 8\}$.

In order to compute the aforementioned metrics, we first uploaded each model to ST boards via the STM Edge AI Developer Cloud tool [103], which allows a seamless import of Artificial Intelligence (AI) models to different tiny-IoT platforms—such as the STM32F469I-DISCO board [102]. Then, through the STM32CubeMX tool [104], a real board may be used by generating the firmware associated with the selected model and flashing it into the selected board. Finally, the performance can be validated through the cloud service by using testing and validation dataset [100]. This has been kept as reference board for the evaluation of the models' complexities, which are shown in Table 4.6 and, for the sake of visual representation, in Figure 4.11 (in terms of n_{MACC}), Figure 4.12 (in terms of RAM), and Figure 4.13 (in terms of t_{exec}), respectively. As

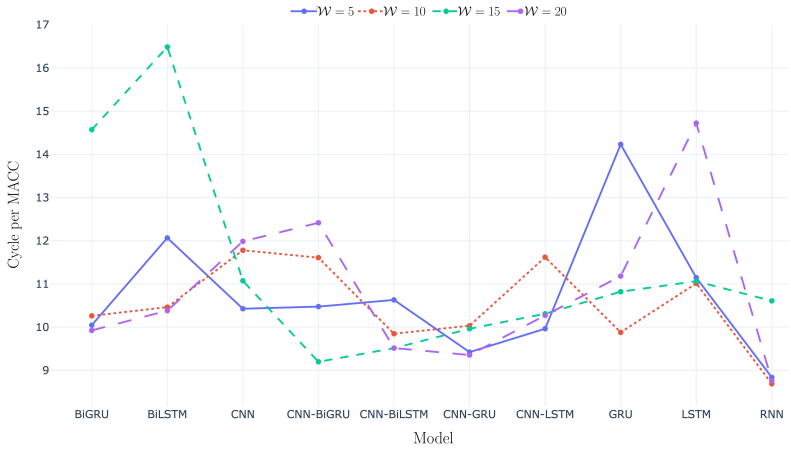


Figure 4.10: Comparison, in terms of number of cycles per MACC, between the considered ML and DL models for different values of the time lag $\mathcal{W}$.

Model	n_{MACC}				RAM (KiB)				t_{exec} (ms)			
	$\mathcal{W}=5$	$\mathcal{W}=10$	$\mathcal{W}=15$	$\mathcal{W}=20$	$\mathcal{W}=5$	$\mathcal{W}=10$	$\mathcal{W}=15$	$\mathcal{W}=20$	$\mathcal{W}=5$	$\mathcal{W}=10$	$\mathcal{W}=15$	$\mathcal{W}=20$
BiGRU	442,047	1,079,656	112,040	2,103,912	6.76	4.2	4.43	7.54	24.67	61.55	9.07	116.00
BiLSTM	389,816	1,425,512	150,440	2,800,232	6.59	7.94	4.66	8.14	26.13	82.86	13.78	161.5
CNN	49,546	112,616	161,256	227,816	4.57	6.14	7.64	8.64	2.87	7.37	9.92	15.17
CNN-BiGRU	185,404	217,546	2,640,104	236,296	6.73	6.69	14.31	7.08	10.79	14.03	134.90	16.30
CNN-BiLSTM	278,526	2,186,216	3,502,056	4,829,416	7.35	12.44	14.93	17.43	16.45	119.6	185.1	255.3
CNN-GRU	267,662	357,585	824,804	1,845,480	7.58	7.99	1094	15.64	14.01	19.93	45.65	95.91
CNN-LSTM	257,256	305,486	1,329,014	2,030,916	7.43	7.36	12.26	15.02	14.24	19.72	76.13	115.9
GRU	20,616	547,688	354,164	522,534	3.45	6.45	5.07	5.27	1.63	30.06	21.29	32.46
LMU	97,892	87,652	112,228	112,228	//	//	//	//	6.11	5.35	6.85	6.85
LSTM	207,374	724,972	1,067,752	286,936	5.85	7.05	7.05	4.70	12.84	44.36	65.62	23.47
TCN	568,400	1,642,676	1,900,544	1,534,100	//	//	//	//	34.73	100.3	116	93.75
RNN	89,424	205,928	164,929	366,008	3.81	4.12	3.92	4.39	4.388	9.94	9.72	17.82

Table 4.6: MACC operations n_{MACC}, RAM usage, and execution time t_{exec} returned by the analyzed ML and DL models, as a function of different time lags $\mathcal{W}$.

can be seen from Table 4.6, among all the models deployable on the chosen IoT board, CNN is the most computationally-efficient (in terms of n_{MACC}) for all time lags, with GRU and RNN ranking as second and third best performing models. At the opposite, CNN-BiLSTM has the higher computational complexity, closely followed by BiLSTM. Then, models with balanced computational efficiency across various time lags include CNN-BiGRU, BiGRU, GRU, and LSTM. Finally, even if LMU and TCN represent novel architectures for time-series prediction (with LMU achieving a high accuracy with a low complexity), for completeness it should be highlighted that they are currently not natively supported by ST Edge AI Developer Cloud [103], a remote tool exploited to run the considered ML and DL models on a remote tiny IoT board directly in the cloud (before being deployed on real devices). Nevertheless, knowing the functional parameters of the chosen IoT board, it has been possible to estimate n_{MACC} and RAM usage also for TCN and LMU.

4.5.4 Post-Training Quantization (PTQ)

Finally, in order to further optimize DL models predicting $PM_{2.5}$ values on tiny IoT devices, an additional investigation has been performed applying a PTQ technique exploiting the TensorFlow Lite (TFLite) library [105]. In detail, by implementing PTQ, a significant decrease (in terms of models' sizes) has been accomplished at the cost of a minimal prediction accuracy decrease. Nevertheless, despite this slight performance reduction, the benefits of reduced model size and improved computational efficiency outweigh the accuracy *trade-off*. In fact, as highlighted in Table 4.7 and Figure 4.14 (where the sizes of original and TFLite-compressed DL models are directly compared), PTQ provides significant benefits in practical situations, especially with resource-constrained systems when computational effectiveness is crucial.

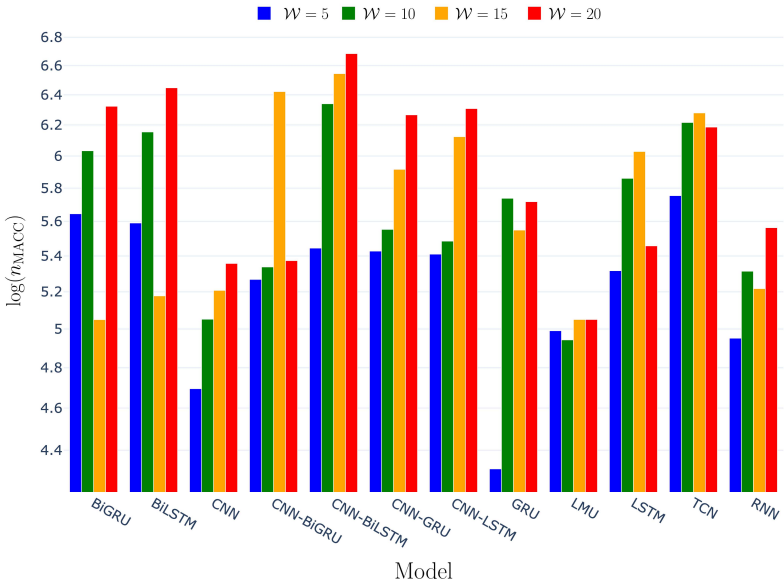


Figure 4.11: Comparison, in terms of n_{MACC} (in logarithmic scale), between the considered ML and DL models for different values of the time lag $\mathcal{W}$.

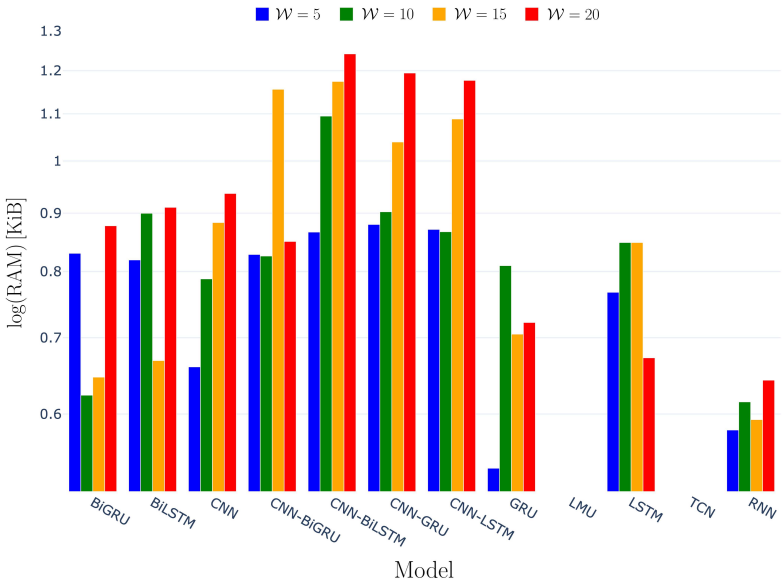


Figure 4.12: Comparison, in terms of RAM (in logarithmic scale), between the considered ML and DL models for different values of the time lag $\mathcal{W}$.

Moreover, Table 4.8 provides a quantitative comparison in terms of average inference time (dimension: [ms]), between the Keras versions of the DL models and their TFLite-compressed counterparts. These results return over 98% reduction in inference time across all evaluated models, confirming the effectiveness of TFLite compression for deployment in resource-constrained environments not only to reduce model size but also to reduce inference time.

As a matter of fact, PTQ with TFLite shows a great potential if applied to efficient and real-time $PM_{2.5}$ monitoring systems, by reducing the computational complexity during model deployment. Finally, it can be noted that reducing the size of the models by means of quantization does not affect significantly the accuracy, as shown in Figure 4.15, where only a (negligible) 1% accuracy reduction is experienced by CNN-BiGRU despite a 66% average model's size's reduction.

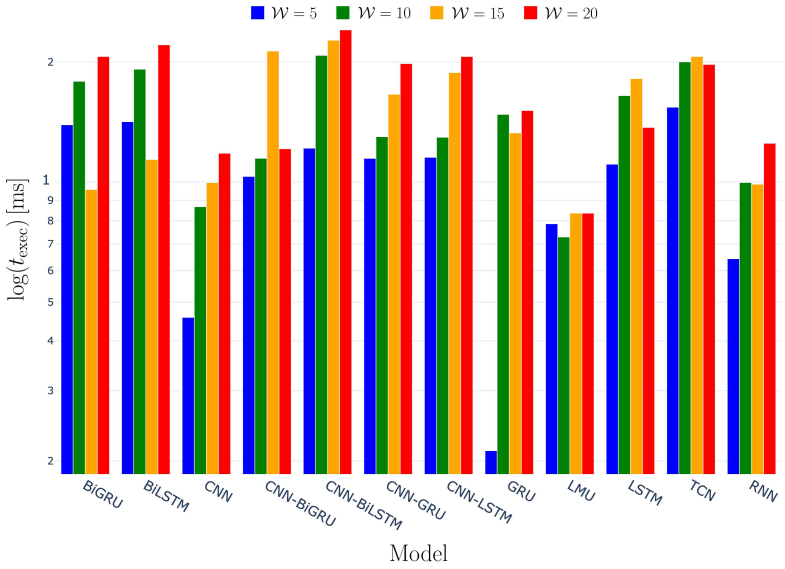


Figure 4.13: Comparison, in terms of t_{exec} (in logarithmic scale), between the considered ML and DL models for different values of the time lag $\mathcal{W}$.

Model	Mean original model size [KiB]	Mean TFLite model size [KiB]	Model size reduction [%]
BiGRU	1347.28	455.86	66.16
BiLSTM	1484.96	499.86	66.34
CNN	1308.85	431.29	67.04
CNN-BiGRU	1157.80	390.12	66.30
CNN-BiLSTM	3143.70	1052.67	66.51
CNN-GRU	1178.98	392.40	66.71
CNN-LSTM	1246.75	414.85	66.72
GRU	551.09	183.82	66.64
LSTM	873.60	291.91	66.58
RNN	537.26	177.76	66.91

Table 4.7: Comparison (in terms of mean size) between original DL models and their corresponding TFLite-compressed versions.

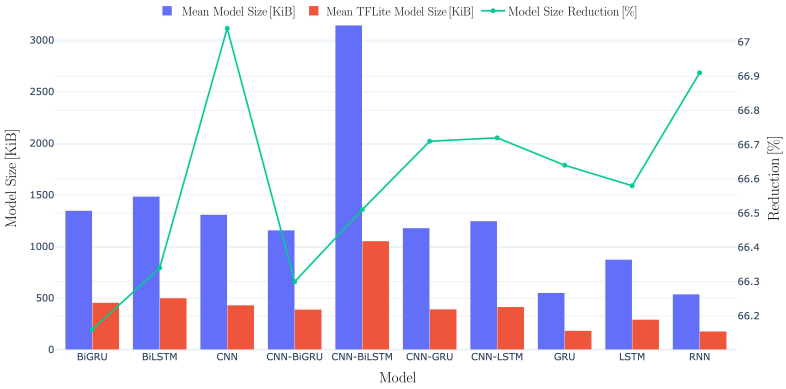


Figure 4.14: Comparison (in terms of mean size) between original DL models and their corresponding TFLite-compressed versions.

Model	Mean original inference time [ms]	Mean TFLite inference time [ms]	Inference time reduction [%]
BiGRU	24.12	0.335	98.61%
BiLSTM	23.10	0.340	98.53%
CNN	24.64	0.015	99.94%
CNN-BiGRU	22.67	0.275	98.79%
CNN-BiLSTM	24.74	0.473	98.09%
CNN-GRU	24.29	0.182	99.25%
CNN-LSTM	22.78	0.192	99.16%
GRU	23.27	0.158	99.32%
LSTM	22.78	0.170	99.25%
RNN	22.66	0.090	99.60%

Table 4.8: Comparison (in terms of inference time) between original DL models and their corresponding TFLite-compressed versions.

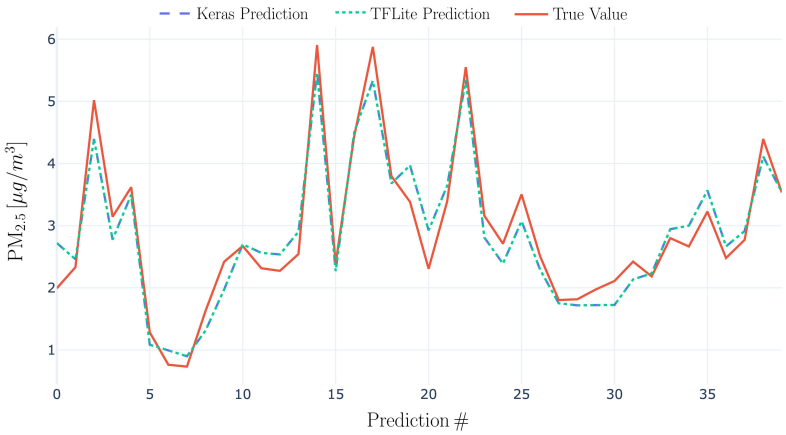


Figure 4.15: Comparison (in terms of prediction accuracy) between true $PM_{2.5}$ values and predicted values obtained through original and quantized CNN-BiGRU models.

4.5.5 Study Limitations

For the sake of clarity and completeness, it is useful to briefly discuss some limitations of the proposed performance analysis.

- A comprehensive evaluation of an optimal sliding window length (as a hyperparameter) would be useful to thoroughly assess the performance of the considered DL models.
- The analysis returned that no single model consistently outperforms the others across all the scenarios considered in our research study.
- A dataset with long-term observations and diverse features would be necessary for training ML and DL models to accurately predict $PM_{2.5}$ levels.

4.6 Conclusions and Future work

This research presents a comparative study of air quality estimation, specifically targeting resource-constrained IoT devices. To this end, several ML and DL models (namely: RNN, CNN, LSTM, GRU, LMU, TCN, BiGRU, BiLSTM, CNN-GRU, CNN-LSTM, CNN-BiGRU, CNN-BiLSTM) have been considered for both short-term and long-term prediction of the $PM_{2.5}$ value, exploiting Bayesian optimization to select the best hyperparameters for model training. Among the analyzed algorithms, CNN-BiGRU shows the highest predictive accuracy and provides an effective trade-off between computational complexity (in terms of number of MACC), memory usage (in terms of

RAM), and execution time. In general, while some DL models may excel in one of the considered performance metrics, they often perform poorly in others. Then, in order to investigate the impact of 8-bit quantization on prediction performance, PTQ has been applied to the considered models: 8-bit quantized CNN-BiGRU achieves approximately a 66% model size reduction with a prediction accuracy reduction equal to only 1% (on average) and a 98.79% inference time reduction. Future research activities might explore the adoption of communication paradigms ensuring data integrity and privacy. Finally, CNN-BiGRU, which is proposed as the best model, can be considered for a real-time $PM_{2.5}$ prediction deployment in particular (e.g., indoor) areas of interest, such as an airport transit area. Additionally, expanding the dataset to include multiple seasons and locations, as well as incorporating heterogeneous datasets, will represent important and interesting future research directions and extensions to further enhance generalizability and external validity of the proposed approach.

Chapter 5

Performance Assessment of ML and DL Models for Network Intrusion Detection on a Constrained IoT Device

5.1 Introduction

The IoT represents a revolutionary technological paradigm where physical objects—denoted as *smart objects* and embedded with sensors, software, and connectivity—seamlessly interact with each other and with end-users across heterogeneous communication networks. Therefore, the IoTenables continuous data collection, sharing, and analysis, sustaining innovative concepts such as *smart cities*, *smart agriculture*, *smart healthcare*, and *smart homes* [106]. In fact, the rapid expansion of IoTis largely driven by advancements in long range wireless networks (e.g., NarrowBand (NB)-IoT and Long Range Wide Area Network (LoRaWAN)), edge computing, SDN, and sensor innovation [107]. As an example, Cisco foresees that, by 2030, the number of IoT-connected devices will be more than 500 B, owing to the increasing global adoption of IoTtechnologies across a variety of industries and sectors [108].

Nevertheless, this rapid growth also brings significant challenges, in particular with regard to scalability, security, privacy, and data management. As the number of connected devices increases, the detection of unauthorized access and malicious activity within IoTnetworks becomes an urgent concern [109], because the widespread deployment of IoTdevices across diverse sectors makes them attractive targets for malicious actors, exposing these devices to a wide range of security threats and vulnerabilities [110].

To this end, security threats on IoT networks vary by network type—literature studies show that approximately 70% of IoT devices exhibit security vulnerabilities, with an average of 25 faults detected per device [111]—while common vulnerabilities may include the following [112]:

- *insecure communication*: data transmission by IoT devices may occur without adequate encryption, enabling attackers to intercept or manipulate communication;
- *lack of device identity management*: the absence of robust Identity Management (IdM) mechanisms complicates the verification of device's authenticity, creating potential security gaps in IoT networks;
- *insufficient authentication and authorization*: many IoT devices lack strong authentication and authorization protocols, allowing unauthorized individuals to access sensitive data or control devices.

To mitigate these risks, Intrusion Detection Systems (IDSs) are recognized as essential components of IoT security infrastructures, monitoring network traffic, detecting potential security threats, and preventing attacks, thereby enhancing the overall security of IoT systems [113]. Therefore, given the growing complexity and scale of IoT networks, the integration of IDSs is crucial to protect against increasingly sophisticated threats targeting connected IoT devices. Simultaneously, applying AI-based models—in particular, ML and DL—can significantly enhance the performance of IDSs, since these models improve the accuracy of detecting and classifying IoT attacks due to their ability to extract patterns of input data.

In this research, the performance of different ML and DL models—namely: simple RNN; LSTM; GRU; MLP; 1D-CNN—on a publicly-available dataset, denoted as CIC IoT2023 [114] and including network traffic traces comprising different attack classes—namely: Denial of Service (DoS); DDoS; Spoofing; Web; Brute Force; Recon; Mirai botnet—are detailed and discussed in terms of accuracy and computational complexity. Moreover, PTQ is considered for compressing pre-trained models for efficient deployment on constrained IoT devices (e.g., STM32H7S78-DK board [115]).

Ref.	Model(s)	FS	DB	HT	IDA	QU
[116]	RNN	✗	✗	✗	✗	✗
[114]	RF	✗	✗	✗	✗	✗
[117]	FL	✗	✓	✗	✗	✗
[118]	LSTM+XGB	✗	✗	✗	✗	✗
[119]	DNN+LGBM	✗	✗	✗	✗	✗
[120]	CNN	✓	✗	✗	✗	✗
[121]	mGRU	✓	✗	✗	✗	✗
[122]	RF	✗	✗	✗	✗	✗
Proposed	**TCN**	✓	✓	✓	✓	✓

FS: Feature Selection; DB: Data Balancing; HT: Hyperparameter Tuning; IDA: IoT Deployability Assessment; QU: Quantization.

Table 5.1: Comparison with related works. Unlike existing literature, the proposed approach integrates model optimization and hardware deployability assessment.

The remainder of the research is organized as follows. In Section 5.2, an overview on related works is given. Section 5.3 discusses a general representation of an IoTsystem, together with an overview on common vulnerabilities. Section 5.4 introduces the chosen dataset, the proposed data analysis techniques, and the considered ML and DL models. Section 5.5 presents the experimental performance results and the PTQ technique. Finally, in Section 5.6 we draw our conclusions.

5.2 Related Works

In order to provide some background on detection and classification of network intrusions and cyber-attacks targeting IoT architectures, in the following we analyze relevant state-of-the-art studies employing ML and DL model architectures and evaluation metrics, providing insights into each model's effectiveness in accurately classifying attacks within the IoT domain. Then, for the sake of clarity and completeness, a comparison between these literature studies and the model proposed in this research, along its main research axes, is shown in Table 5.1.

In [116], the performance of several DL models—namely: DNNs [123], CNNs, and RNNs—is discussed considering their application to a publicly-available dataset (namely, CIC IoT2023 [114]). The experimental results show that RNNs return the best performance among all the considered models—with an accuracy of 96.52%, a precision of 96.25%, a recall of 96.52%, and an F1-score of 95.73%. This reflects the effectiveness of RNNs in detecting both normal and malicious packets, outperforming the other evaluated DL models.

In [114], the authors evaluate five different ML methods—namely: Logistic Regression, Perceptron, AdaBoost, RF, and DNN—across three classification tasks—namely, binary, 8-class, and 34-class classification. Among them, RF demonstrates the highest performance with 83% recall, 70% precision, and 71% F1-score for the 34-class classification, and 91% recall, 70% precision, and 71% F1-score for the 8-class classification. Finally, RF excels in binary classification with 96% recall, precision, and F1-score, demonstrating its robust performance across varying classification complexities.

In [117], a Federated Learning (FL)-based IDS, aiming at enhancing cybersecurity in IoT networks, is proposed, with FL achieving a 99.0% accuracy in detecting binary attacks and showcasing the potential of FL in distributed environments where data privacy is essential. Then, class imbalance in the training dataset—a common issue in intrusion detection tasks—is addressed applying the Synthetic Minority Oversampling Technique (SMOTE) [124].

A hybrid model, combining LSTM networks with XGBoost to improve the detection of Mirai botnet attacks, is proposed in [118], benchmarking this approach against alternative models—including DNNs, CNNs, standalone LSTM, and a hybrid RF-LSTM model. Obtained experimental results show that LSTM-XGBoost outperforms the others, in detail achieving a 97.7% accuracy and 97.4% precision, recall, and F1-score, thus highlighting its ability to detect complex botnet patterns with high precision and reliability. Additionally, Mirai detection is explored in [119], where a DNN-Light Gradient Boosting Machine (LGBM) hybrid model achieves strong results with accuracy, precision, recall, and F1-score up to 95%. This performance significantly exceeds

that of other models, such as SVM, LGBM, and Stochastic Gradient Descent (SGD), which reach 71%, 85%, and 52% accuracy scores, respectively.

In [120], a CNN-based model is proposed for attack detection, featuring a 4.14 MB model size and a 6 s inference time. This CNN model outperforms alternative models (DNN and LSTM) for binary classification, showing precision, recall, and F1-score equal to 99.43%, 99.40%, and 99.41%, respectively.

A DL-based DDoS detection mechanism is introduced in [121] using a modified GRU (mGRU) model optimized for low computational complexity. In detail, mGRU delivers strong performance metrics, achieving 98% precision, recall, and F1-score. Nevertheless, its efficiency is limited by a high execution time (equal to 10.49 s), suggesting potential improvements in real-time applications.

Finally, binary and multi-class detection of network attacks using classifiers such as SVM, AdaBoost, DT, and RF, are discussed in [122]. In detail, AdaBoost delivers strong results in binary classification, returning a 96% recall, a 95% precision, and a 96% F1-score. However, the larger the number of classes, the more significant the performance degradation: 8-class classification returns recall, precision, and F1-score drops to 48%, 56%, and 48%, respectively, while 34-class classification reflects 50% recall, 58% precision, and 49% F1-score.

5.2.1 TinyML for Intrusion Detection

In [125], the authors investigate the efficiency of TinyML for IoT security by quantizing a two-layer LSTM model (with 64 and 32 units) into a TinyLSTM using TensorFlow Lite on the CICEVSE2024 dataset [126]. As a result, the quantized model achieves an F1-score of up to 99.83%, while reducing the baseline model size of LSTM by 82%, the inference time by 76.5%, and the inference memory by 96.89%.

Authors in [127] demonstrate the critical trade-offs in deploying TinyML for IoT cybersecurity, showing that while dynamic and static 16-bit floating point quantization techniques effectively balance performance (preserving over 85% accuracy on the USTC-TFC2016 dataset [128] while halving model sizes and accelerating inference), more aggressive 8-bit integer quantization achieves maximum compression (up to 75% size reduction) at the cost, however, of severe accuracy degradation. On the other hand, PQT maintains a high detection rate but sacrifices latency and efficiency gains, which are essential for real-time edge deployment.

Finally, in [129] a TinyML model for IoT attack detection, combining a lightweight DT and a small NN with a Logistic Regression meta-learner, is deployed on an Arduino Nano 33 BLE Sense. The model achieves exceptional performance on the ToN-IoT dataset [130], returning 99.98% accuracy, 99.94% F1-score, and a near-perfect false positive rate equal to 0.0009, while operating with ultra-low latency (0.12 ms) and power consumption (0.01 mW).

5.3 Vulnerabilities in IoT Architectures

5.3.1 IoT Layers

Before analyzing in detail the suitability of the considered AI models in supporting on IDS in identifying and classifying IoT network attacks, it is useful to discuss on the general architecture of an IoT system, as shown in Fig. 5.1. In detail, an IoT node can be seen as composed of four layers, each featuring the following specific responsibilities in contributing to its seamless integration with other devices.

- *Perception Layer:* this layer, also known as *sensing layer*, corresponds to the lowest layer in an IoT architecture and consists of physical devices—such as sensors, actuators, and smart appliances—directly interacting with the environment to collect and process real-time data.
- *Network Layer:* inside an IoT domain, this layer manages the connectivity and the data transmission between IoT devices, gateways, and the higher layers, thus exploiting an ensemble of communication protocols (e.g., Zigbee, BLE, IEEE 802.11 Wi-Fi, LTE/5G cellular, LoRaWAN) to support seamless information transfer across the IoT ecosystem.

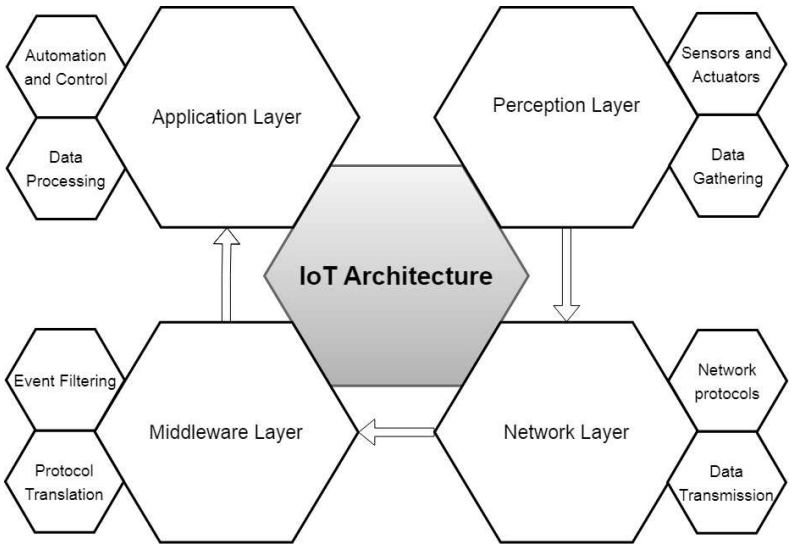


Figure 5.1: General mapping of an IoT architecture.

- *Middleware Layer:* this layer acts as an intermediary layer between *network* and *application layers* and provides essential services such as data filtering, aggregation, and protocol translation, thus ensuring interoperability and easing efficient management and deployment of IoT applications.
- *Application Layer:* this layer enables specific use cases by delivering services and insights to end-users and additional applications, in detail directly interfacing with the users and providing an ecosystem for the deployment of IoT applications (e.g., smart homes, healthcare, and industrial automation).

5.3.2 IoT Security Vulnerabilities

In the following, a brief discussion on well-known vulnerabilities of IoT systems is provided from different perspectives.

5.3.2.1 Active and Passive Attacks

Knowing that cyber-attacks affecting IoT networks can differ with respect to the specific targeted network, a first classification can focus on the *type of actor* conducting the attack, namely active or passive.

- *Active attack:* it refers to a type of security attack wherein the attacker engages in *direct* communication with the intended target system or network. This attack involves the deliberate alteration or disruption of a network's operations through the injection of malicious traffic or the execution of unauthorized commands.
- *Passive attack:* it is classified as a security attack wherein the attacker establishes an *indirect* connection with the target network and observes the communication occurring therein. In this case, an assailant would observe, intercept, or surreptitiously listen to data transfers without making any modifications or exerting any influence over the information, thus primarily aiming to illegally obtain access to sensitive or secret data or information without raising suspicion or detection.

For completeness, in Subsection 5.3.2.2, Subsection 5.3.2.3, Subsection 5.3.2.4, and Subsection 5.3.2.5, we delve into attacks specific for each IoT layer detailed in Subsection 5.3.1, namely: perception layer, network layer, middleware layer, application layer, respectively. A comprehensive summary of these attacks is further detailed in Table 5.2.

Layer	Attack Type	Description
Perception Layer	Tampering	Physically accessing and controlling IoThardware or altering settings.
	DoS	Overwhelming IoT sensors with traffic, causing data loss and service disruption.
	Jamming	Interfering with wireless communication (e.g., Wi-Fi, Bluetooth), blocking data transmission.
	Fake node injection	Adding unauthorized nodes to spread false information across the network.
	Sleep deprivation	Keeping sensor nodes active to deplete battery power and disrupt network operations.
Network Layer	Routing attacks	Altering network paths to redirect data to malicious nodes or disrupt communication.
	IP spoofing	Using falsified IP addresses to impersonate trusted devices.
	Sybil attack	Creating fake identities to generate incorrect data and disrupt network functionality.
	Traffic analysis	Intercepting data packets to extract valuable information.
Middleware Layer	MitM	Intercepting and potentially manipulating communication between IoT devices and middleware.
	Malware	Using malicious software like viruses or Trojans to access sensitive information.
	DoS	Overloading the network with data, consuming resources and preventing legitimate access.
Application Layer	XSS	Embedding malicious scripts into an IoT platform interface, compromising data and device functionality.
	SQL injection	Executing unauthorized SQL commands by altering input data in database queries.
	Sniffing attack	Monitoring data traffic between IoT devices and application services without permission.

Table 5.2: Network attacks categorized along their main targeted architecture layer.

5.3.2.2 Attacks to the Perception Layer

The main cyber-challenges the IoT perception layer might suffer for can be summarized as follows.

- *Tampering:* this attack aims at creating a direct physical connection with the IoT device and controlling its operational tasks, then involving hardware manipulation (e.g., establishing connections with ports or interfaces on the device) or modifications to its settings [131].
- *DoS:* this attack foresees an attacker flooding the IoT sensors with an extraordinary volume of traffic or requests, leading to sensor overload, data loss, and service disruption [132].
- *Jamming:* this corresponds to a deliberate attack strategy disrupting or obstructing wireless communication signals, thus preventing information transmission [133].
- *Fake node injection:* this is one of the most severe attacks for an IoT device, since it features a malicious node to be injected into an IoT network looking for gaining access to it, then spreading misleading information across the network itself. Consequently, unauthorized nodes will enable an attacker to access the entire network.
- *Sleep deprivation attack:* this attack aims at affecting the energy source of (often battery-powered) sensor nodes. In fact, this attack keeps IoT nodes *active* by altering their sleep cycle—usually employed to minimize their energy consumption—and, consequently, their functionalities, quickly depleting and disrupting the overall IoT network [134].

5.3.2.3 Attacks to the Network Layer

The main cyber-challenges affecting the network layer can be summarized as follows.

- *Routing attacks:* this attack focuses on altering the routes established in the IoT network, thus aiming at transmitting data and messages to malicious intermediary nodes, or interfering with the network's data transmission process.
- *IP Spoofing:* this attacks features a malicious node posing as another device or changing the origin IP address inside the packets flowing inside the network itself, to tamper the IoTnetwork and gain an unauthorized access.
- *Sybil attack:* this attack foresees that the attacker will try to access the network by using a phony identity, in turn also activating fake nodes as normal devices within the network and impairing the network's operation by creating incorrect and/or large amount of information [135].
- *Traffic analysis attack:* this attack targets to analyze and intercept data packets exchanged *intra* IoT nodes, as well as between IoTdevices and remote entities, in order to extract valuable information.

5.3.2.4 Attacks to the Middleware Layer

The main cyber-challenges the middleware layer might suffer from can be summarized as follows.

- *Man-in-the-Middle (MitM) attack:* this attack foresees the ability of an unauthorized entity to intercept (and potentially manipulate) the communication occurring between IoT devices.
- *Malwares:* threats like viruses, Trojan horses, and malwares represent a (sub-)set of tools used by attackers to access (in an unauthorized way) undisclosed and private information, with data theft happening exploiting executable codes (often developed in machine language).
- *DoS attack:* this class of intrusion foresees an attacker overwhelming the IoT network with a heavy amount of requests, thus resulting in the network's congestion due to excessive traffic. As a result, IoT devices deplete energy and resources, thus preventing the user from accessing his/her data.

5.3.2.5 Attacks to the Application Layer

Finally, the main cyber-challenges affecting the application layer can be summarized as follows.

- *Cross-Site Scripting (XSS):* in this type of attack, the attacker adds harmful scripts to be run (at run-time) into the IoT node's command-line or Web interfaces, thus compromising its responsiveness as well as the user's protected information.
- *SQL injection:* this attack occurs when the attacker succeeds in manipulating the input parameters used in a SQL query and in executing such malicious SQL queries into the target database.
- *Sniffing attack:* this attack involves the ability of the attacker to successfully collect and monitor the data traffic between IoT devices and the services at the application layer, thus handling and controlling them without any authorized permission.

5.4 Proposed Model

In the following, the proposed IDS, designed to accurately identify and classify IoT attacks by means of AI models suitable for deployment on constrained IoT nodes, is discussed. The proposed models (detailed in Section 1.5) demonstrate their ability to discover intricate interconnections among features in the considered dataset. For completeness, as further outlined in Subsection 5.4.1.3, we selectively omit the features which do not significantly contribute to the classification process, thus obtaining an efficient training process execution by diminishing the complexity without sacrificing essential correlations.

Dataset			Samples No.
2 Classes	**8 Classes**	**34 Classes**	
Malicious	**DDoS**	ACK_Fragmentation	23,365
		HTTP_Flood	2,360
		ICMP_Flood	586,613
		ICMP_Fragmentation	36,833
		PSHACK_Flood	333,880
		RSTFINFlood	329,190
		SYN_Flood	330,999
		SlowLoris	1,928
		SynonymousIP_Flood	292,280
		TCP_Flood	367,051
		UDP_Flood	440,357
		UDP_Fragmentation	23,504
	DoS	HTTP_Flood	5,930
		SYN_Flood	163,947
		TCP_Flood	217,539
		UDP_Flood	270,629
	Mirai	greeth_flood	80,162
		greip_flood	61,335
		udpplain	72,657
	Spoofing	DNS_Spoofing	14,662
		MITM-ArpSpoofing	25,175
	Recon	HostDiscovery	10,868
		OSScan	8,172
		PingSweep	149
		PortScan	6,640
		VulnerabilityScan	3,059
	Web	SQLInjection	429
		CommandInjection	404
		BrowserHijacking	453
		XSS	302
		Uploading_Attack	105
		Backdoor_Malware	283
	BruteForce	DictionaryBruteForce	1,045
Benign			89,695

Table 5.3: Classification of network attacks and their occurrences, divided by their major class.

5.4.1 Dataset

In this study, the CICIoT2023 dataset [114] comprising 46,686,579 records collected from 105 IoT devices, has been identified as reference dataset. More in detail, the traffic traces include 1,098,195 *benign* records and 45,588,384 *malicious* records, and encompasses 33 types of cyber-attacks, such as DoS, DDoS, spoofing, brute force, and Mirai Botnet attacks.

Then, as further detailed in Table 5.3, three variations of the dataset have been considered, in order to conduct a comprehensive evaluation: (i) a 34-class ($\mathcal{C} = 34$) dataset retaining all the attack types; (ii) an 8-class ($\mathcal{C} = 8$) dataset, obtained by grouping together similar classes of attack; and (iii) a binary (2-class, $\mathcal{C} = 2$) dataset, obtained by splitting the network attacks into *benign* and *malicious* predominant classes. As a *first* remark, this allows for a thorough analysis of the performance returned by the proposed IDS exploiting different DL models. Then, as a *second* remark, reducing the number of classes enhances the approach generalization, decreases the training time, and lowers memory and computational requirements.

Looking at the distributions of the considered attack classes detailed in Table 5.3, it should be highlighted that the majority of the occurrences are related to DoS and DDoS attacks, thus leading to a priori imbalanced dataset. This aspect has to be carefully taken into account when discussing

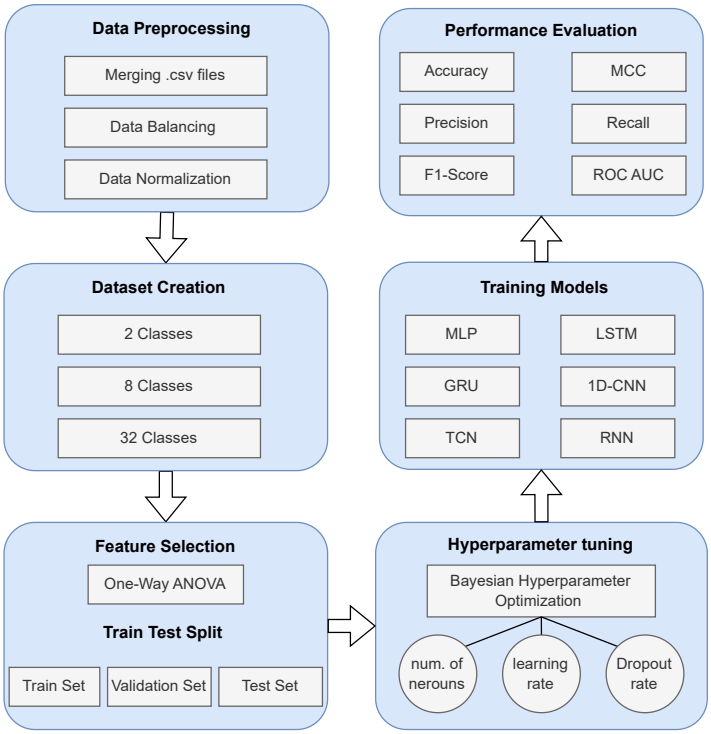


Figure 5.2: Data workflow and model definition and evaluation.

the performance of the AI models embedded in an IDS.

Finally, even for clarity and completeness, the overall approach adopted to refine the dataset—addressing its imbalance and complexity—and to train the chosen AI models, is shown in Fig. 5.2. Then, each AI-based algorithm is subsequently evaluated exploiting an ensemble of metrics as detailed in Subsection 5.4.3.

5.4.1.1 Data Balancing

In order to address the imbalance of the dataset (as mentioned in Subsection 5.4.1) and mitigate the overfitting effect, the target's sample size has been defined as based on the data labeled as *benign* and representing normal network flows. Furthermore, a Random UnderSampler [136] has been applied in order to reduce the majority classes to match the target sample number—equal to 878,556—and to ensure a more balanced class distribution without introducing bias toward dominant categories. Then, a Random OverSampler [124] has been applied to the minority classes to increase their sample size, thus aligning them with the target. This combined approach enhances the ability of the model to generalize across both *benign* and *malicious* instances, thus improving the performance of the IDS while reducing the risk of false positives. Moreover, balancing the dataset before training prevents the model from learning the dominant class distribution, thus ensuring a more comprehensive understanding of all categories. The combined application of these sampling techniques addresses the challenge of oversampling a heavily imbalanced dataset, which may lead to extremely long training time and the risk of generating non-representative samples, possibly causing a degradation in the model's ability to generalize.

5.4.1.2 Data Normalization

Since a normalization stage is essential for ensuring all features to uniformly contribute to the model (preventing any single variable from dominating the learning process), they are measured

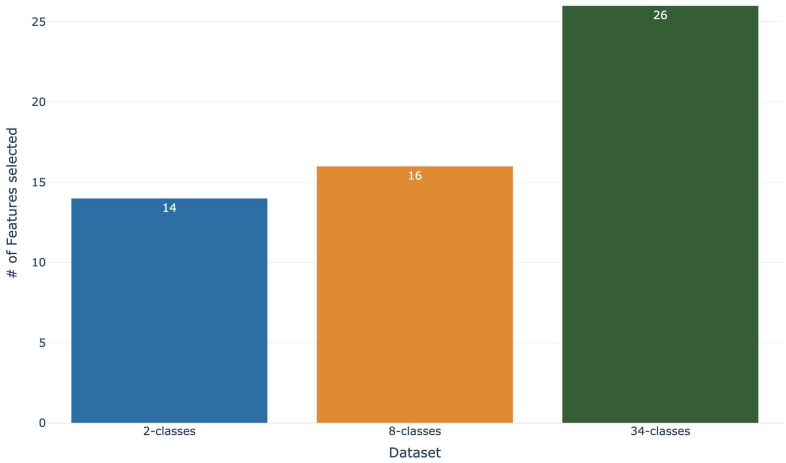


Figure 5.3: One-way ANOVA feature selection.

on the chosen CIC IoT2023 dataset on different scales. Unfortunately, this may negatively affect the models' performance. Hence, using a scaler is crucial to mitigate the impact of these varying scales, leading to a fair consideration of each feature during model training.

To this end, a PowerTransformer [137], not only normalizing the features but also stabilizing the variance and making the data distribution more Gaussian-like, has been employed. This transformation is particularly beneficial, as the model's convergence speed increases and its overall performance improves, thus allowing more effective learning and generalization.

5.4.1.3 Feature Selection

Finally, particular attention has been dedicated to the quality of the dataset. Knowing that quality plays a critical role in the performance of ML/DL models, since datasets often contain features with low variance that contribute little to the model's ability to capture underlying patterns, various feature selection techniques can be considered to retain only the most informative features.

Once the features are standardized to a uniform scale, different methods can be applied to rank their relevance. In this study, the one-way ANOVA approach [138] is employed to assess and rank each feature based on its significance, since ANOVA quantifies the extent to which a feature differentiates between the means of distinct classes, assigning higher scores to features exhibiting higher discriminatory capabilities.

Moreover, to further refine the selection, a threshold to identify a significant drop in feature importance between consecutive features has been applied. In particular, a 30% drop in the ANOVA score between two adjacent features has been set as the *cut-off* to distinguish highly relevant features from those with a limited impact. Nevertheless, the selection of a minimum number of features is generated.

Overall, this approach was applied to all three versions of the dataset (2-classes, 8-classes, and 34-classes), each characterized by distinct distributions of *benign* instances. Consequently, feature selection was performed separately for each version, with many features consistently identified as top contributors across all three datasets. As shown in Fig. 5.3, increasing the number of classes requires to retain additional features to effectively correlate them with specific classes. This impacts both size and complexity of the trained models, as a larger feature set can potentially result in a simpler model for classifying the attack.

5.4.2 HyperParameter Optimization (HPO)

With specific regard to DL models, it is well-known that hyperparameters, although remaining fixed during the training process, yet play a crucial role in determining the model's architecture and the overall performance efficiency. Consequently, an automatic tuning of categorical, discrete, and continuous hyperparameters is accomplished through an HPO task. More in detail, the primary

Model Type		Hyperparamater	Range
Generic Parameter		Learning Rate	[1e−4, 1e−2]
		Batch Normalization	[True, False]
		Activation Function	[tanh, relu]
Model-specific Parameter	RNN	# Layers	[1, 2] (step=1)
		Layer Units	[64, 512] (step=16)
		Dropout Value	[0.2, 0.4] (step=0.1)
		Dense Units	[324, 256] (step=16)
	CNN	Convolutional Filters	[32, 256] (step=32)
		Kernel Size	[3, 5, 7]
		Pool Size	[2, 4] (step=1)
		Dilatation Rate	[2, 4, 8]

Table 5.4: Hyperparameters considered for the model tuning, along with their defined ranges.

goal of HPO is to identify the optimal hyperparameters set minimizing the chosen performance metric analytically defined as:

$$x^{\star} = \arg\min_{x \in X} f(x) \tag{5.1}$$

where: $x^{\star}$ represents the optimal hyperparameters configuration; and $f(x)$ corresponds to the model's performance metric, evaluated on a validation set, to be minimized.

Moreover, we adopt a BO for hyperparameter tuning: this effectively enhances the performance of the evaluated DL models by systematically exploring the hyperparameters' space and balancing both exploration and exploitation. On the operational side, the KerasTuner library [139] has been employed to effectively implement BO. For the sake of completeness, Table 5.4 presents the optimized parameters along with their defined ranges in the search space.

5.4.3 Evaluation Metrics

In order to assess the performance of the chosen DL models on the selected dataset for the IDS of interest, the following evaluation metrics have been employed.

5.4.3.1 Accuracy

Accuracy (denoted as $\mathcal{A}$) measures the ratio between the number of correctly predicted instances and the total number of instances. It can be expressed as

$$\mathcal{A} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

The accuracy $\mathcal{A}$ ranges from 0 (with none of the instances being correctly classified) to 1 (with all instances being correctly classified). Although $\mathcal{A}$ is an indicator of the overall model performance, it may be misleading in the case of imbalanced datasets.

5.4.3.2 Precision

Precision (denoted as $\mathcal{P}$)indicates the ratio between the number of correctly predicted positive cases and the total number of positives. It can be expressed as

$$\mathcal{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}\,.$$

The precision $\mathcal{P}$ ranges from 0 (worst case) to 1 (best case). Consequently, a high precision reduces the false positive rate, making it critical in applications where false positives are critical.

5.4.3.3 Recall

Recall (denoted as $\mathcal{R}$ and also as *sensitivity*) measures how effectively the model allows to detect truly positive instances. It can be defined as the ratio between the number of true positives and the sum between the number of true positives and false negatives (i.e., the number of all positives):

$$\mathcal{R} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

The recall $\mathcal{R}$ ranges from 0 (worst case) to 1 (best case). Therefore, a high recall is essential when missing positive cases is detrimental, such as in medical diagnoses or intrusions detection.

5.4.3.4 F1-Score

F1-Score (denoted as $\mathcal{F}$) balances precision $\mathcal{P}$ and recall $\mathcal{R}$ as follows:

$$\begin{aligned} \mathcal{F} &= 2 \times \frac{\mathcal{P} \times \mathcal{R}}{\mathcal{P} + \mathcal{R}} \\ &= \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}. \end{aligned} \tag{5.2}$$

The F1-Score $\mathcal{F}$ ranges from 0 (if the precision $\mathcal{P}$ or the recall $\mathcal{R}$ is 0) to 1 (indicating perfect precision and recall). It is useful especially for imbalanced datasets, as it simultaneously takes into account both false positives and false negatives.

5.4.3.5 Matthews Correlation Coefficient (MCC)

It is defined as follows:

$$\mathcal{M} = \frac{(\text{TP} \times \text{TN}) - (\text{FP} \times \text{FN})}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}.$$

Unlike the F1-Score, MCC offers a more balanced perspective on the performance by incorporating all confusion matrix elements, with values ranging from -1 (no prediction at all) to $+1$ (perfect prediction). In other words, the best case is with $|\mathcal{M}|$ close to 1.

Unlike accuracy, MCCis more robust in the case of imbalanced datasets.

5.4.3.6 Cohen's Kappa Coefficient

The Cohen's Kappa Coefficient [140] (denoted as κ) is a statistical measure quantifying the agreement between *predicted* and *actual* labels, while adjusting for agreement occurring by chance. It is defined as follows:

$$\kappa = \frac{2 \times (\text{TP} \times \text{TN} - \text{FN} \times \text{FP})}{(\text{TP} + \text{FP}) \times (\text{FP} + \text{TN}) + (\text{TP} + \text{FN}) \times (\text{FN} + \text{TN})}.$$

Unlike accuracy, κ provides a more reliable evaluation in the presence of class imbalance. In fact, κ values range from -1 (complete disagreement) to $+1$ (perfect agreement), where 0 indicates an agreement on a random chance. In other words, the best case is with $|\kappa|$ close to 1.

5.4.3.7 Area Under the Curve Receiver Operating Characteristic (AUCROC)

The Receiver Operating Characteristic (ROC) curve $r(x)$ is a function of the threshold x: it returns the accuracy of a binary classifier model deciding on the basis of the threshold x. It corresponds to the ratio between true positives and false positives. The AUC can be expressed as follow:

$$\text{AUC} = \int_0^1 r(x)\, \mathrm{d}x$$

where $r(x)$ is the ROC curve. It represents the model's capability to distinguish between classes. In fact, a perfect classifier has an AUC equal to 1, while an AUC equal to 0.5 denotes a limited discrimination ability.

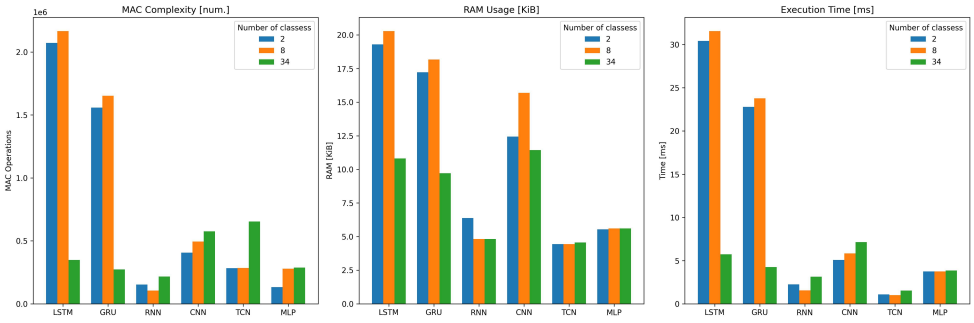


Figure 5.4: Comparison, in terms of number of MACC, RAM usage, and execution time, between the considered FP32 DL models.

Model	MACCs [$\times 10^3$]			RAM [KiB]			Exec. Time [ms]		
	C2	C8	C34	C2	C8	C34	C2	C8	C34
LSTM	2072.0	2166.6	348.4	19.3	20.3	10.8	30.4	31.6	5.7
GRU	1558.3	1651.9	273.7	17.2	18.2	9.7	22.8	23.8	4.2
RNN	152.9	105.8	217.5	6.4	4.8	4.8	2.2	1.6	3.1
CNN	406.1	494.8	575.9	12.4	15.7	11.4	5.1	5.8	7.1
TCN	282.8	284.8	653.5	4.4	4.4	4.6	1.1	1.0	1.5
MLP	132.8	278.9	288.4	5.5	5.6	5.6	3.7	3.7	3.8

Table 5.5: Computational complexity (MACCs, RAM usage, and execution time) of the considered FP32 DL models across different class configurations ($\mathcal{C}$).

5.4.4 Model Complexity

In the following, the computational complexity of the considered models is evaluated in terms of three evaluation metrics, namely: MACC, RAM usage, and execution time. The evaluation has been conducted through the STMicroelectronics Cube.AI Analyzer tool [115], a cloud-based platform designed to assess the performance of ML/DL models on resource-constrained devices, with 32-bit Floating Point (FP32) models being deployed on a STM32H7S78-DK board [115] featuring a 600 MHz ARM Cortex-M7 microcontroller, 620 KB internal memory, and 16 MB external RAM.

The obtained experimental results, shown in Fig. 5.4 and detailed in Table 5.5 for clarity and completeness, indicate LSTM as the model with the highest computational demand and requiring substantially more MACC and RAM than the other algorithms. In contrast, TCN and MLP exhibit the lowest average computational complexity, with TCN achieving a 94.61% reduction in the execution time (if compared to LSTM) and using approximately 73% less RAM than GRU. Similarly, MLP reduces the number of MACC operations by 84.73% in comparison to LSTM. This makes both TCN and MLP suitable for deployment on resource-constrained devices where computational efficiency is fundamental.

Finally, it should be highlighted that the considered DL models have been evaluated not only on the basis of their accuracy in detecting several types of attacks, but also considering their deployability according to multiple computational metrics. To this end, given the challenges of deploying complex AI models on resource-constrained devices, 8-bit INTeger (INT8) quantization of activations and weights has been applied to further reduce the computational complexity and enhance the efficiency. A detailed discussion of these quantization techniques and their impact on the performance is presented in Subsection 5.5.2.

5.5 Results

In order to present the findings of the proposed experimental analysis, detailing the performance of each considered DL model, we *first* analyze each model's ability to accurately classify network traffic, distinguishing between benign and malicious network activities. *Then*, we examine the

Model	$\mathcal{A}$	$\mathcal{F}$	$\mathcal{P}$	$\mathcal{R}$	κ	$\mathcal{M}$	ROC-AUC
LSTM	**0.9937**	**0.9897**	**0.9923**	**0.9886**	**0.7990**	**0.8155**	**0.9986**
GRU	**0.9937**	0.9896	**0.9923**	0.9885	0.7983	0.8150	0.9985
RNN	0.9931	0.9890	0.9919	0.9877	0.7868	0.8052	0.9984
CNN	0.9928	0.9884	0.9916	0.9871	0.7780	0.7977	0.9984
TCN	0.9932	0.9889	0.9919	0.9877	0.7863	0.8047	0.9984
MLP	0.9932	0.9888	0.9918	0.9875	0.7837	0.8026	0.9985

Table 5.6: Performance of considered FP32 DL models in the 2-class ($\mathcal{C} = 2$) anomaly detection.

Model	$\mathcal{A}$	$\mathcal{F}$	$\mathcal{P}$	$\mathcal{R}$	κ	$\mathcal{M}$	ROC-AUC
LSTM	0.8807	0.8830	0.8860	0.8807	0.7291	0.7292	**0.9802**
GRU	**0.8867**	**0.8877**	**0.8896**	**0.8867**	**0.7391**	**0.7392**	0.9799
RNN	0.8566	0.8630	0.8731	0.8566	0.6862	0.6888	0.9790
CNN	0.8643	0.8692	0.8761	0.8643	0.6977	0.6987	0.9790
TCN	0.8761	0.8795	0.8841	0.8761	0.7190	0.7191	0.9796
MLP	0.8800	0.8823	0.8855	0.8800	0.7268	0.7268	0.9798

Table 5.7: Performance of considered FP32 DL models in the 8-class ($\mathcal{C} = 8$) anomaly detection.

Model	$\mathcal{A}$	$\mathcal{F}$	$\mathcal{P}$	$\mathcal{R}$	κ	$\mathcal{M}$	ROC-AUC
LSTM	0.8674	0.8682	**0.8897**	0.8674	0.8547	0.8564	**0.9952**
GRU	0.8604	0.8587	0.8831	0.8604	0.8469	0.8485	0.9946
RNN	0.8074	0.8033	0.8697	0.8074	0.7900	0.7973	0.9947
CNN	0.8571	0.8584	0.8816	0.8571	0.8434	0.8451	0.9949
TCN	0.8594	0.8602	0.8856	0.8594	0.8460	0.8481	0.9950
MLP	**0.8687**	**0.8694**	0.8895	**0.8687**	**0.8561**	**0.8577**	0.9948

Table 5.8: Performance of considered FP32 DL models in the 34-class ($\mathcal{C} = 34$) anomaly detection.

models' performance across different classes and levels of complexity, looking for the optimal configuration suitable for real-time IoT security applications.

5.5.1 Network Traffic Classification Performance

As shown in Table 5.6, LSTM achieves the highest performance in the context of 2-class ($\mathcal{C} = 2$) anomaly detection, providing the highest accuracy (99.37%) and showing high values of the other evaluation metrics. Instead, as shown in Table 5.7, GRU demonstrates slightly better results in the context of 8-class ($\mathcal{C} = 8$) anomaly detection, outperforming the other models in terms of accuracy $\mathcal{A}$, precision $\mathcal{P}$, recall $\mathcal{R}$, F1-score $\mathcal{F}$, κ, and MCC $\mathcal{M}$. Finally, as detailed in Table 5.8, considering the more complex 34-class ($\mathcal{C} = 34$) anomaly detection task, MLP emerges as the top-performing model, achieving the highest accuracy, F1-score, κ, MCC, and recall. Nevertheless, the differences (in terms of performance) returned by the models are minimal, showing that no single model consistently dominates across all the tasks. Therefore, computational efficiency and complexity trade-offs should guide the model's selection, in particular with regard to applications requiring a deployment on resource-constrained devices or real-time processing environments.

5.5.2 Post-Training Quantization (PTQ)

Since (as mentioned in Section 5.4) it is of interest to verify the deployability of the considered DL models on resource-constrained IoT devices, the application of a PTQ technique can emerge as a highly effective mechanism to reduce the computational complexity and enhance the efficiency. In detail, PTQ specifically focuses on compressing pre-trained models for efficient deployment on *tiny* IoT devices by converting models' weights from `FP32` values to `INT8` values, thus significantly reducing both model size and memory footprint and enhancing inference efficiency without requiring a re-training. On the operational side, we employ both Keras and TFLite libraries, with PTQ compressing and quantizing the Keras model by converting it to the TFLite

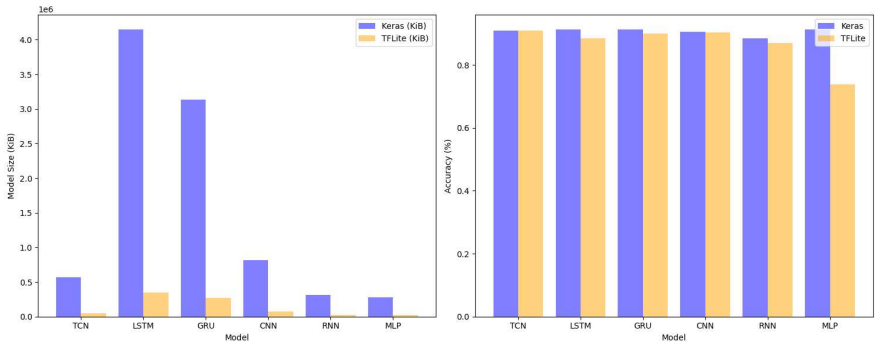


Figure 5.5: Comparison, in terms of model size and accuracy, of the considered DL models using both Keras and TFLite.

format, which, in turn, corresponds to a lightweight data representation useful for deploying ML and DL models on resource-constrained embedded devices [141].

After applying PTQ on the considered AI models, the obtained results are shown in Fig. 5.5, where an average model size's reduction of 91.24%, with an average accuracy drop of only 4% across all considered anomaly detection classes, is shown. This highlights the significance of quantization to optimize resource utilization and energy efficiency in constrained devices. In fact, quantizing CNNs to an 8-bit precision usually results in a slight performance degradation with the benefit of a significant memory utilization reduction [142].

5.5.3 Accuracy-Computational Complexity Trade-Off

In order to simultaneously evaluate both accuracy and computational complexity of the considered DL models, we define a new *trade-off score*, denoted as ψ, to provide a balanced assessment by considering normalized accuracy alongside computational complexity metrics, including the number of Multiply and ACCumulates (MACCs), RAM usage, and execution time [143]. In the following, we summarize the steps required to calculate ψ. We remind that ψ is an effective evaluation metric in relative terms, i.e., it allows to fairly compare a set of DL models mong themselves. However, ψ does not represent an "absolute" performance metric of a single DL model.

5.5.3.1 Metrics Normalization

All the considered metrics are normalized to the range $[0, 1]$ using the following min-max scaling strategy:

$$\vartheta_k^{(\text{norm})} = \frac{\vartheta_k - \vartheta_k^{(\text{min})}}{\vartheta_k^{(\text{max})} - \vartheta_k^{(\text{min})}} \tag{5.3}$$

where: $k \in \{\text{"macc"}, \text{"ram"}, \text{"itime"}\}$ corresponds to the considered evaluation metric (namely: number of MACCs, RAM usage, inference time, respectively); ϑ_k represents the original value of the k-th metric; $\vartheta_k^{(\text{min})}$ and $\vartheta_k^{(\text{max})}$ correspond to the minimum and maximum values of ϑ_k across all considered DL models; $\vartheta_k^{(\text{norm})}$ is the normalized value of the k-th metric.

5.5.3.2 Efficiency Score Computation

In order to represent the computational complexity of each considered model, we introduce an *efficiency score*, denoted as ξ, defined as the following weighed average of the normalized complexity metrics (defined in Subsection 5.5.3.1):

$$\xi \triangleq w_{\text{macc}} \cdot \vartheta_{\text{macc}}^{(\text{norm})} + w_{\text{ram}} \cdot \vartheta_{\text{ram}}^{(\text{norm})} + w_{\text{itime}} \cdot \vartheta_{\text{itime}}^{(\text{norm})} \tag{5.4}$$

where: w_{macc}, w_{ram}, and w_{itime} represent the weights assigned to each complexity metric, respectively.

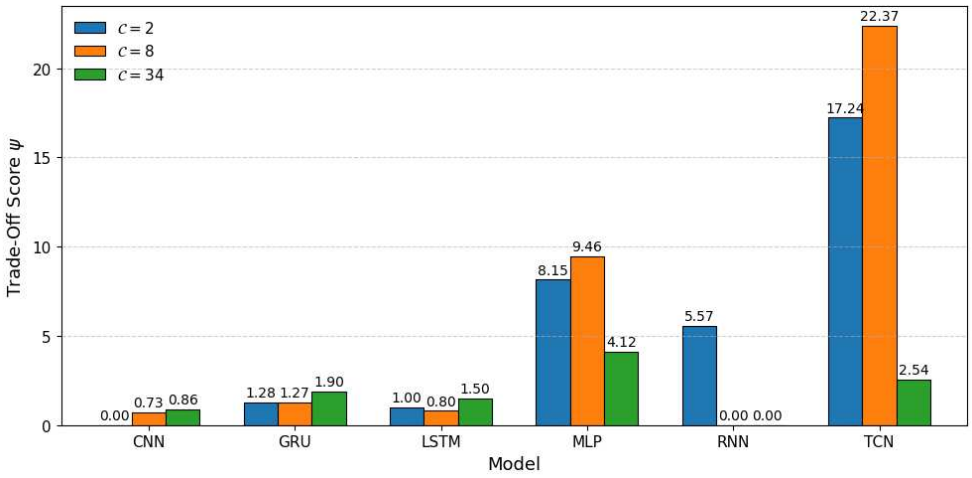


Figure 5.6: Trade-off score ψ obtained by the considered DL models across the different 2-, 8-, and 34-class datasets.

In order to avoid a biased performance analysis, we assign the same weight to all considered evaluation metrics, i.e., we set $w_{\text{macc}} = w_{\text{ram}} = w_{\text{itime}} = 1/3$.

5.5.3.3 Trade-Off Score

Finally, the trade-off score ψ, balancing the ratio between normalized accuracy $\vartheta_{\mathcal{A}}^{(\text{norm})}$ and efficiency score ξ, can be defined as follows:

$$\psi \triangleq \frac{\vartheta_{\mathcal{A}}^{(\text{norm})}}{\xi}. \tag{5.5}$$

According to the definition in (5.5), a high value of ψ is representative of a better performance jointly considering accuracy and computational resource utilization.

For completeness, the trade-off score ψ obtained by the considered DL models across the different datasets is shown in Figure 5.6. As can be observed, TCN emerges as the algorithm offering the most efficient balance between accuracy and computational complexity in the 2- and 8-class cases, i.e., in simple intrusion detection scenarios. Instead, in the more complex 34-class scenario, MLP is the algorithm returning the best trade-off.

5.5.4 Deployment on Resource-Constrained Devices

In order to evaluate the practical feasibility of deploying DL models on resource-constrained devices, quantized TCN and MLP models—being the best performing models, according to the results shown in Subsection 5.5.3.3—were implemented and tested on the STM32H7S78-DK board, trying to assess how the number of output classes affects TCN and MLP's computational and memory requirements when executed directly on the target hardware.

More in detail, the performance was analyzed in terms of number of MACCs, RAM usage, and inference time, with these metrics obtained through *on-board* profiling tools and real-time measurements during the inference phase on the STM32H7S78-DK board. The obtained metrics are shown in Table 5.9, where quantized `INT8` TCN and MLP models have been considered for the different output class configurations of interest—namely, for $\mathcal{C} \in \{2, 8, 34\}$.

For the sake of completeness, the relative difference (denoted as $\Delta\vartheta$) between the performance returned by `FP32` and quantized `INT8` TCN and MLP models, considering the different output class configurations (namely, $\mathcal{C} \in \{2, 8, 34\}$), is shown in Table 5.10. In particular, this percentage difference is calculated as follows:

$$\Delta\vartheta_k = \frac{\vartheta_k^{(\text{int8})} - \vartheta_k^{(\text{fp32})}}{\vartheta_k^{(\text{fp32})}} \tag{5.6}$$

Model	$\mathcal{A}$			MACCs [$\times 10^3$]			RAM [KiB]			Inf. Time [ms]		
	C2	C8	C34	C2	C8	C34	C2	C8	C34	C2	C8	C34
TCN	0.993	0.782	0.701	293.1	295.2	674.8	18.5	18.5	31.5	3.57	3.77	8.07
MLP	0.993	0.652	0.487	132.0	277.9	287.3	7.7	8.8	8.8	1.48	2.92	3.06

Table 5.9: Performance and complexity of quantized INT8 TCN and MLP models: Accuracy ($\mathcal{A}$), MACCs, RAM usage, and inference time across different class configurations (C).

Model	Δ MACC			Δ RAM			Δ Inf. Time			Δ Flash			Δ Size		
	C2	C8	C34	C2	C8	C34	C2	C8	C34	C2	C8	C34	C2	C8	C34
TCN	+3.6	+3.6	+3.2	+308	+308	+574	+225	+269	+424	+1.3	+1.1	-0.1	-90.8	-90.8	-91.1
MLP	-0.6	-0.3	-0.3	+36.2	+52.5	+52.5	-16.3	-21.2	-19.8	-71.6	-73.2	-73.2	-84.3	-90.7	-80.2

Table 5.10: Relative difference (Δ%) between FP32 and INT8 models (TCN and MLP) across MACCs, RAM, inference time, flash, and model size for different class configurations (C).

where: $k \in \{\mathcal{A}, \text{“macc”}, \text{“ram”}, \text{“itime”}, \text{“flash”}, \text{“msize”}\}$ corresponds to the considered evaluation metrics (namely, accuracy, number of MACCs, RAM usage, inference time, flash usage, model size, respectively); $\vartheta_k^{(\text{fp32})}$ and $\vartheta_k^{(\text{int8})}$ represent the original value of the k-th metric returned by `FP32` and quantized `INT8` DL models, respectively.[1]

As highlighted in Table 5.10, `INT8` quantization produces a significant model size reduction for both TCN and MLP, exceeding 90% in most cases. In particular, with regard to MLP, the number of MACCs remains essentially consistent (with variations less than 1%), the RAM usage increases moderately (between +36% and +53%), the inference time decreases (between −16% and −21%), and the flash usage is significantly reduced (between −71% and −73%), thus reflecting lower weights storage needs. This highlights how dense layers (e.g., MLP) benefit from reduced computational cost and memory requirements (when quantized), thus leading to faster inference despite slightly higher RAM demands.

In contrast, TCN provides a different result. While model size is also reduced by more than 90% and flash usage remains nearly constant, RAM usage and inference time of quantized versions increase substantially—between +309% and +574%, and between +225% and +425%, respectively. This behavior primarily occurs because TCN architectures, in particular one-dimensional convolutional layers, cannot be fully quantized to `INT8`, while, instead, MLP supports complete `INT8` quantization. This is particularly evident given the type of MACCs shown in Table 5.11:

1. `smul_f32_f32` performs a scalar multiplication between two `FP32` values;
2. `op_f32_f32` represents a generic operation—such as addition, activation, or normalization—applied to `FP32` data;
3. `smul_s8_s8` and `op_s8_s8` perform, adopting signed `INT8` data types, the same operations operated by MACCs 1 and 2;
4. `smul_s8_f32` and `smul_f32_s8` perform conversions from signed `INT8` to `FP32` and vice versa.

Thus, it is clear how TCN limitations in `INT8` quantization arise because of an increased internal complexity—in terms of additional layers and MACC operations, mainly dilated convolutions and dynamic padding—being not supported in integer form by both TensorFlow Lite and Cube-AI [144] from STMicroelectronics [145].

This leads to an unexpected increase in both RAM and flash memory usage, since several conversion and intermediate operations (to bridge between `FP32` and `INT8` arithmetic) are required during the quantization task. More in detail:

- as indicated by the operations' breakdown, the bulk of the computation still relies on `FP32` multiplications (namely, `smul_f32_f32`), with several conversion layers having to be

[1] For better readability, instead of including the values of flash memory occupation and model size in Table 5.5 (for `FP32` DL models) and Table 5.9 (for `INT8` DL models), they have been included in Table 5.11 (columns 5–8).

Model	C	MACC				Flash Size [MB]		Model Size [MB]	
		FP32		INT8		FP32	INT8	FP32	INT8
TCN	2	smul_f32_f32	280,546	smul_s8_s8	290	1.08	1.10	3.25	0.29
		op_f32_f32	2,270	smul_s8_f32	10,812				
				smul_f32_f32	280,256				
				smul_f32_s8	1,024				
				op_s8_s8	30,734				
TCN	8	smul_f32_f32	282,472	smul_s8_s8	872	1.09	1.10	3.28	0.29
		op_f32_f32	2,360	smul_s8_f32	10,848				
				smul_f32_f32	281,600				
				smul_f32_s8	1,024				
				op_s8_s8	824				
TCN	34	smul_f32_f32	650,626	smul_s8_s8	6,754	2.50	2.49	7.49	0.65
		op_f32_f32	2,846	smul_s8_f32	21,684				
				smul_f32_f32	643,872				
				smul_f32_s8	1,216				
				op_s8_s8	1,310				
MLP	2	smul_f32_f32	131,938	smul_s8_s8	131,938	0.51	0.15	1.60	0.15
		op_f32_f32	862	op_s8_s8	30				
MLP	8	smul_f32_f32	277,784	smul_s8_s8	277,784	1.07	0.29	3.25	0.29
		op_f32_f32	1,160	op_s8_s8	120				
MLP	34	smul_f32_f32	286,834	smul_s8_s8	286,834	1.10	0.30	3.35	0.30
		op_f32_f32	1,550	op_s8_s8	510				

Table 5.11: Detail on the type and number of MACCs, flash and model size returned by FP32 and quantized INT8 TCN and MLP models.

added *before* and *after* convolutional and element-wise operations (e.g., `smul_s8_f32` and `smul_f32_s8`);

- each additional layer requires temporary buffers to store intermediate activations in both `FP32` and and `INT8` format: this drastically increases the RAM consumption at runtime and, in the (worst) case the device features insufficient internal RAM temporary buffers should be stored into external RAM, much more degrading the performance;
- each quantized layer needs to store its scale and zero-point parameters for proper rescaling between quantized tensors: this results in the need to add metadata that partially negate the expected flash memory reduction and, similarly to the RAM discussion, in the case of insufficient internal flash memory, external one should be used, this worsen again the performance.

The final consequence is that, even though quantization decreases the overall weight storage by about 90%, the proliferation of intermediate buffers and conversion operations results in higher runtime memory demands and slightly larger flash usage—especially in architectures extensively using convolutional and residual connections, such as TCN.

Overall, the obtained results highlight that `INT8`quantization effectively reduces model and memory requirements for embedded deployment, while its impact on RAM and inference time depends on the specific model architecture.

Comparing the accuracy metric of `FP32` and `INT8` format of MLP and TCN models for all classes, it can be seen from Table 5.10 that accuracy loss due to quantization is negligible in binary anomaly detection but increases with more classes. It has also been observed that TCN models are more resilient to quantization than Multi-Layer Perceptrons (MLPs), further confirming their suitability for resource-constrained devices.

5.5.5 Quantized Models Accuracy-Computational Complexity Trade-Off

Finally, we evaluate the performance of quantized TCN and MLP models on the anomaly detection task using the trade-off score ψ. For the 2-class task, both TCN and MLP models achieve high accuracy (above 99%), with TCN slightly outperforming MLP, albeit with higher resource usage due to mixed-precision quantization. Then, as the task complexity increases (with 8-class and 34-class configurations), TCN maintains significantly higher accuracy and trade-off score, whereas the fully quantized MLP sacrifices accuracy in exchange for reduced model size and complexity.

5.6 Conclusions

In this study, several DL models—namely, LSTM, GRU, RNN, CNN, TCN, and MLP—have been evaluated over the CICIoT2023 dataset for anomaly detection on resource-constrained IoT devices. In order to optimize the hyperparameters of the considered DL models, BO has been applied, with DL models being evaluated for both binary (2-class) and multi-class (8-class, 34-class) intrusion detection tasks. Then, the models' performance has been assessed on a STM32H7S78-DK board (virtually available through the cloud-based STM Cube.AI Analyzer tool) on the basis of computational complexity metrics such as number of MACCs, RAM usage, and inference time. In order to evaluate the deployability of these DL models on resource-constrained IoT devices, a trade-off score ψ, balancing both classification accuracy and computational complexity, has been introduced. Our results show that TCN and MLP guarantee the most efficient balance with respect to the other DL models, on the basis of the specific intrusion detection task. Furthermore, PTQ has been considered in order to evaluate the impact, on both model size and classification accuracy, of parameter quantization from `FP32` to `INT8`. The obtained experimental results demonstrate a significant model size reduction (greater than 80%) for both TCN and MLP with all considered classes. However, there are differences between TCN and MLP: models based on dense layers (such as MLP) benefit from lower computational costs and memory requirements after quantization; convolution-based models (such as TCN), instead, exhibit increased runtime and RAM usage—particularly when the number of output classes grows—while showing only a marginal increase in the number of MACCs. These findings highlight that post-training `INT8` quantization should be applied by taking into account the considered model and the final goal (e.g., latency or computational complexity).

Chapter 6

DL-Aided Intrusion Detection Systems Performance Comparison in SDN-Oriented Networks

6.1 Introduction

The IoT comprises a wide range of physical or virtual devices with diverse capabilities, all interconnected over the Internet. The number of connected IoT devices is estimated to exceed 32.1 B by 2030 [146]. In recent years, IoT has been deployed in heterogeneous scenarios, including smart cities [147], [148], precision agriculture [149], [150], remote patient monitoring [151], [152], intelligent homes [153], [154], and connected transportation systems [155], [156]. These applications leverage the ability of IoT in collecting and analyzing real-time data, enabling informed decision-making and processes automation.

This rapid expansion of Internet-connected devices has created a pressing need for innovative networking technologies able to enhance resource management and scalability. To this end, the Software-Defined Networking (SDN) paradigm revolutionizes networking management by decoupling the *control plane* from the *data plane* and providing greater flexibility and efficiency in traffic management (i.e., via traffic engineering approaches [157], [158]). In fact, unlike traditionally managed networks, where data and control planes are tightly coupled, SDN allows for programmable policies automating traffic management and resource allocation, then resulting in optimized bandwidth usage and reduced latency. This innovation avoids traditional, device-specific control mechanisms in routers and switches, thus leading to more flexible, scalable, and efficient networks. From an architectural point of view, the SDN is composed of three layers, namely (i) *application layer*, (ii) *control layer*, and (iii) *data layer* [159], [160], with a communication between these layers often enabled via the OpenFlow protocol [161].

Despite many advantages offered by SDN with respect to traditional networks, its centralized control approach might appear as a single point of failure. This creates a need for robust IDSs ensuring network security [162] through real-time network traffic monitoring (analyzing both "genuine" and "anomalous" packets), thus acting as an early-warning warning mechanism and enabling immediate decision-making. To this end, AI models might represent effective solutions for intrusion detection in SDN environments, with DL mechanisms—based on architectures like MLP, RNN, and CNN—capable of processing real-time data to detect malicious activities within a network. Moreover, supporting multi-class anomaly detections, DL models enable the application of different rules and policies based on nature and attributes of each attack.

In this work, different DL models—namely: MLP, LSTM, GRU, BiLSTM, BiGRU, CNN, CNN-LSTM, CNN-GRU, CNN-BiLSTM, CNN-BiGRU—are evaluated for multi-class intrusion detection on traffic traces contained in the public InSDN dataset [163] and collected in an SDN-based scenario. The performance of the selected algorithms is evaluated considering different performance metrics, namely: accuracy, complexity, and F1-Score. Furthermore, the deployability of the considered DL models on *tiny* IoT devices, characterized by limited and constrained resources, is assessed analyzing how full and dynamic `INT8` quantization might influence both resource optimization and detection accuracy. For this reason, alternative compression mechanisms (such as WP) are also evaluated.

The remainder of the paper is organized as follows. In Section 6.2, an overview on IoT and SDN paradigms, as well as on the considered DL models, is presented. Section 6.3 briefly presents different related works. Section 6.4 introduces the chosen dataset and the proposed data processing and analysis techniques. In Section 6.5, the experimental performance is investigated considering also the computational complexity. Finally, in Section 6.6 we draw our conclusions.

6.2 Background

6.2.1 The IoT Paradigm

As the IoT has evolved in the last years, different architectures have been proposed in order to meet the growing requirements highlighted by both the application scenarios and end-users. To this end, one of the common models used to described the IoT is the *three-layer model*, consisting of *perception layer*, *network layer*, and *application layer* [164], [165].

1. *Perception Layer*: this layer is composed of sensors and actuators collecting data from the environment. Therefore, it plays a crucial role in enabling context awareness, a key feature of IoT, by gathering real-time information.
2. *Network Layer*: this layer transmits the collected data to the *application layer*, thus ensuring that sensor data reach the processing systems.
3. *Application Layer*: this layer provides the services and interfaces useful—sometimes needed—for various IoT applications, including, for example: smart homes, smart agriculture, and smart cities. At this layer, data are ultimately used to deliver value to end-users.

Nevertheless, while the three-layer architecture might be simple yet foundational, the rapid expansion and the increasing complexity of IoT systems have highlighted its limitations. In order to overcome these limitations, a more advanced *six-layer architecture*—extending the aforementioned three preliminary layers with the following three additional layers—has been defined to better manage data flow, processing, and user interaction as follow [166].

4. *Transport Layer*: this layer is responsible for transmitting the data from *application layer* to *processing layer* exploiting various communication technologies—such as 3G, 4G, WiFi, and RFID. So as, this layer acts as a *bridge*, ensuring that data will be securely and efficiently moved across the architecture.
5. *Processing Layer*: at this layer, data received from the *transport layer* are stored, computed, and processed,. This layer handles data management and ensures that information will be ready for analysis and decision-making.
6. *Visualization Layer*: this layer focuses on data visualization and interpretation for the end-users, in particular exploiting flowcharts, graphs, and dashboards, in order to support an informed decision-making.

6.2.2 The Software-Defined Networking (SDN) Paradigm

As shown in Figure 6.1, an SDN-based architecture can be modeled as based on three planes, namely: *data*, *control*, and *application*. More in detail, the *data plane* consists of network devices—e.g., switches and routers—responsible for forwarding data on the basis of predefined rules managed by the *control plane*, that, in turn, ensures packet forwarding to the intended destination. Hence, the data plane receives the incoming packets and forwards them according to routing or flow table entries supplied by the control plane.

The *control plane*, "logically centralized" in the SDN controller, orchestrates the network traffic by applying routing and resource-allocation decisions. It communicates with the data plane via specific protocols (such as OpenFlow [167]) through Application Programming Interface (API) denoted as *southbound API*. The SDN controller also maintains a global view of the network topology, simplifying the packet management across the network. IT also includes the Network Operating System (NOS) which manages the network resources and provides communication services. Unfortunately, the fact (as mentioned before) that the SDN's core functionality is logically centralized in the SDN controller might introduce a single point of failure, making it vulnerable to attacks such as Distributed Denial of Service (DDoS), with the result that these kinds of attacks can overwhelm the SDN controller and disrupt network operations.

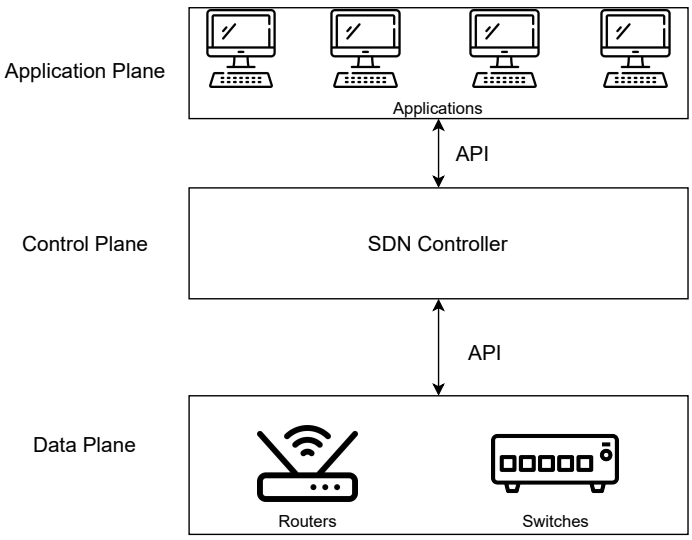


Figure 6.1: Pictorial representation of an SDN-based architecture.

Finally, the *application plane* is located at the top of the SDN architecture and acts as an enabler for a wide range of applications, including security, monitoring, and traffic management. In detail, it interacts with the control plane through specific API denoted as *northbound API*. This plane facilitates the programmability of physical devices within the network, enabling a dynamic control over the network's behavior.

As a general consideration, this clear layered separation of duties not only simplifies network management, but it also enhances its flexibility, scalability, and overall responsiveness to changing demands. As a matter of fact, it is noteworthy to highlight that an SDN architecture can significantly vary on the basis of different factors, including network scale, performance requirements, security considerations, and the need of integration with legacy systems. Additionally, this flexible architecture can also be implemented on older traditional systems, enabling organizations to overlay SDN functionalities on existing infrastructure and to gradually move to more modern and agile network environments [168].

6.2.3 Deep Learning (DL) Models

in the following section, an overview of the DL models considered in this work is presented. In addition to the LSTM, GRU, and CNN architectures discussed in Section 1.5, several additional architectures are explored for this task. Furthermore, hybrid configurations combining convolutional and recurrent components, such as CNN-GRU, CNN-LSTM, CNN-BiGRU and CNN-BiLSTM are also taken into account to leverage both spatial and temporal feature representations.

6.2.3.1 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP) is a type of NN consisting of multiple layers of neurons using non-linear activation functions to capture complex patterns. Moreover, an MLP typically comprises three main components: (i) the *input layer*, receiving the input data and having a number of neurons equal to the number of features in the considered dataset; (ii) the *hidden layers*, consisting of one or more layers of neurons enabling the NN to learn and extract complex patterns from the input data; and (iii) the *output layer*, producing the final predictions and containing a specific number of neurons for each specific task (e.g., one neuron for binary classification, multiple neurons for multi-class classification). The computation performed by a single neuron can be

mathematically expressed as follows:

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{6.1}$$

where: n corresponds to the number of neurons, x_i represents the i-th input; w_i is the i-th corresponding weight; b is the bias term; $\phi(\cdot)$ denotes the activation function.

6.2.3.2 Bidirectional LSTM (BiLSTM

Bidirection LSTM (BiLSTM) is an extension of LSTM networks that processes data in both forward and backward directions. By processing the input sequence in two directions, BiLSTM can capture information from both the past and the future context of a sequence.

6.2.3.3 Bidirectional GRU (BiGRU)

Bidirection GRU (BiGRU) works similarly to BiLSTM but with GRU units instead of LSTM units. Like BiLSTM, BiGRU processes input sequences in both directions, thus capturing dependencies in both forward and backward directions of the sequence.

6.3 Related Works

For the sake of completeness, before diving into the experimental performance analysis of the DL models detailed in Section 1.5, an overview on related literature works focusing on similar aspects is useful for the reader.

An LSTM-based IDS is presented in [169] to distinguish between attack traffic and benign traffic. The proposed model comprises 500 hidden units arranged in four LSTM layers, achieving 99.73% accuracy, 98.6% detection rate, and 98.9% precision for the InSDN dataset (lated detailed in Subsection 6.4.1).

A DNN-based DDoS detection model, consisting of four hidden layers (with 128, 256, and 128 neurons, respectively), trained for 30 epochs with the ADAM optimizer [170], and applied on SDN traffic, is proposed in [171], showing its ability to achieve a 99.98% accuracy and a loss of 0.0052%.

In [172], three different models, including DNN, CNN, and LSTM, are evaluated. According to the obtained results, CNN outperforms the others, providing an accuracy rate of 99%, while DNN and LSTM achieve 87% and 90% accuracy, respectively.

In [173], various ML models—including DT, Naive Bayes, Logistic Regression, Stochastic Gradient Descent, and MLP—are evaluated for binary malicious detection. The authors further evaluate the influence of different feature extraction methods—such as RF, Recursive Feature Elimination (RFE), and Chi-square—to improve the detection accuracy. According to the obtained results, DT provides the highest accuracy (99.13%).

An LSTM-based AE model for DDoS detection in SDN networks is presented in [174], with the model consisting of three hidden layers (with 32, 16, and 8 neurons, respectively). Then, to enhance the classification accuracy, the authors employ Information Gain (IG) and RF for feature selection, making the proposed model achieve a 99.93% detection precision with 48 used features, 99.90% with the 10 most relevant features selected by the RF algorithm, and 99.96% when using 10 features selected by IG.

A CNN-BiLSTM model is proposed in [175] for binary and multi-class attack classification in an SDN environment. The approach incorporates an RF classifier for feature selection and applies random oversampling to balance the dataset. The results demonstrate that the proposed model achieves an 97.77% accuracy, 97.51% recall, 99.85% precision, and an F1-score of 95.28% for binary classification. For multi-class attack classification, the model attains an average performance of 97.1%.

A Convolutional Neural Network with Attention (CNNA)-BiLSTM is proposed for binary and multi-class classification in [176], where a Genetic Algorithm (GA) is employed to select the most significant features. The proposed model achieves an accuracy of 99.89% for binary

Ref.	Method	Data Balancing	Feature Selection	HPO	Quantization	IoT Development
[169]	LSTM	✗	✗	✗	✗	✗
[171]	DNN	✗	✗	✗	✗	✗
[172]	CNN	✓	✗	✗	✗	✗
[173]	DT	✓	✓	✗	✗	✗
[174]	LSTM AE	✗	✓	✗	✗	✗
[175]	CNN–BiLSTM	✓	✓	✓	✗	✗
[176]	Attention-CNN–BiLSTM	✗	✓	✗	✗	✗
Proposed Model	CNN-GRU	✓	✓	✓	✓	✓

Table 6.1: Comparison of related studies and the proposed model across key evaluation criteria.

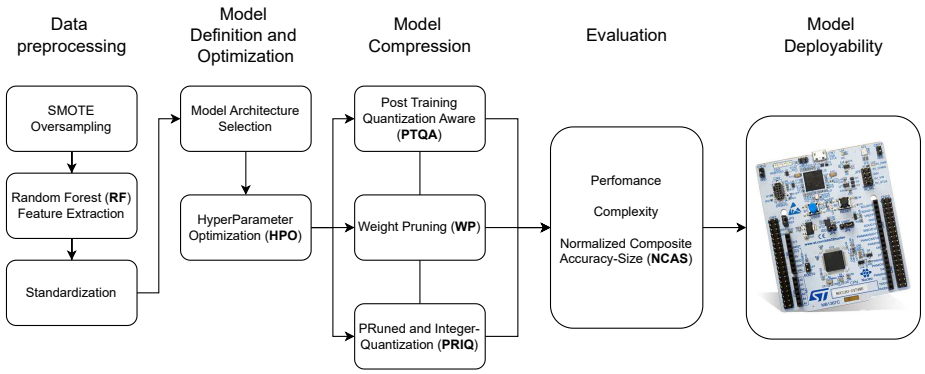


Figure 6.2: General representation of the workflow defined in the proposed IDS's performance evaluation.

attack detection and 99.31% for multi-class attack detection, thus being comparable with LSTM, CNN-LSTM, and CNN-BiLSTM models, and demonstrating superior performance.

6.4 Proposed Methodology

In the following, the proposed model, based on PRuning and Integer Quantization (PRIQ) and shown in Figure 6.2), is introduced and compared with Post-Training Quantization Aware (PTQA) and WP, then providing experimental results considering accuracy and complexity as performance metrics. Moreover, a discussion on how the models have been trained, as well as on the methodology used for HPO, is presented.

6.4.1 Dataset

As anticipated in Section 6.1, the InSDN dataset [163] is used as the reference dataset in this work. In particular, InSDN was developed to evaluate IDSs in SDN environments, and generated using a virtualized testbed including several Virtual Machines (VMs), an OpenFlow switch, and an SDN controller: together, these elements simulate a realistic SDN environment. The dataset captures both benign and malicious traffic classes. In the second class, the dataset includes a wide array of cyber-attacks such as DoS, DDoS, brute-force attacks, WEB application exploitation, probing, and botnet activities. Depending on the target devices and traffic characteristics, the InSDN dataset can be divided into three categories: (i) normal traffic, (iii) attacks directed at a vulnerable Metasploitable 2 Virtual Machine (VM) [177], and (iii) attack traffic targeting the SDN controller or the underlying OVS [178]. These three categories are briefly described as follow.

- **Normal traffic**: contains benign traffic consisting of 68,424 data records.
- **Metasploitable 2 traffic**: includes 136,743 attack traffic records, particularly targeting the vulnerable Metasploitable 2 VM.

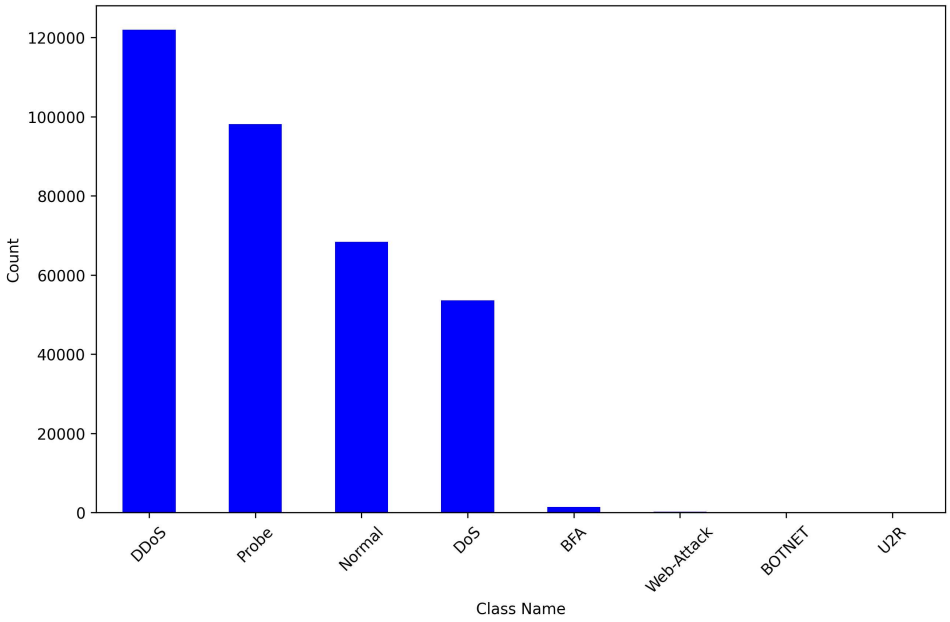


Figure 6.3: Class distribution in the merged dataset.

Table 6.2: Attack distribution in Metasploitable 2 and OVS groups.

Attack Type	Records Count	
	Metasploitable 2	OVS
DoS	1,145	52,471
DDoS	73,529	48,413
Probe	61,757	36,372
Brute-Force Attack	295	1,110
Exploitation (R2L)	17	–
WEB Attack	–	192
Botnet	–	164

- **OVS traffic**: comprises attack traffic directed at the OVS switch and SDN environment and includes 138,722 data records.

In the proposed research study, a total of 343,889 records (returned by the joint utilization of the above three categories) have been used, with an uneven distribution—shown in Figure 6.3—among the various types of attack—described in Table 6.2.

6.4.1.1 Dataset Classes

For the sake of completeness, we underline that the combined dataset consists of eight distinct classes, each representing different types of network malice, as follows.

- **DoS attack**: aims at overwhelming network resources, making services unavailable to legitimate users by flooding the network with malicious traffic.
- **DDoS attack**: similar to a DoS attack, but simultaneously "fired" by multiple malicious systems, this amplifying the volume of traffic and the impact on the target system.
- **Probe attack**: scans the network to gather information about available services, open ports, or vulnerabilities, often as a precursor to more damaging intrusions.
- **Brute-Force attack**: systematically attempts all possible combinations of passwords or encryption keys to gain unauthorized access to systems.

Class Label	Encoded Value	Before Resampling	After Resampling
BFA	0	984	85,359
BOTNET	1	115	85,359
DDoS	2	85,359	85,359
DoS	3	37,531	85,359
Normal	4	47,897	85,359
Probe	5	68,690	85,359
U2R	6	12	85,359
WEB-Attack	7	134	85,359

Table 6.3: Label encoding mapping regarding the dataset classes (left), and class distribution *before* and *after* data resampling (right).

- **Web attacks**: exploit vulnerabilities in WEB-based applications, such as SQL injection, XSS, or file inclusion, aiming to compromise data or control the server.
- **Botnet attack**: corresponds to a collection of infected devices controlled remotely by an attacker, often used to conduct large-scale attacks such as DDoS or spam campaigns, as well as to spread malware.
- **Exploitation (R2L) attacks**: external attacks where the attacker tries to gain unauthorized local access to a target machine over the network by exploiting network or software vulnerabilities.

6.4.1.2 InSDN Network Architecture

The experimental testbed where the InSDN dataset was collected consists of four VMs deployed using VMware Workstation on a Windows 10 host system, as follows.

- **Attacker Machine**: this was a Kali Linux VM used to simulate cyber-attacks.
- **SDN Controller**: this was an Ubuntu 16.04 VM running the ONOS controller [179], responsible for managing the SDN network.
- **SDN Switch and Mininet**: this was embodied by another Ubuntu 16.04 VM hosting Mininet [180] and the OVS switch, them both enabling network emulation.
- **Target Server**: this was a Linux-based Metasploitable 2 VM providing exploitable services for testing various attack vectors.

6.4.2 Data Preprocessing

In order to process the InSDN dataset—detailed in Subsection 6.4.1—through DL models, the following preprocessing techniques must be applied to extract only the meaningful information and provide a balanced training set.

6.4.2.1 Label Encoding

Given the fact that the InSDN dataset includes different types of traffic data (as detailed in Subsection 6.4.1.1), a label encoder [181] is applied to assign a unique numerical value to each class, as shown in Table 6.3 (first two columns).

6.4.2.2 Training and Test Subset Splitting

Then, in order to prepare the merged dataset for the experimental performance evaluation of the considered DL models (in terms of training and performance evaluation), a 70%–30% splitting—with 70% of the data being used for training and 30% for testing purposes—is applied.

6.4.2.3 Oversampling

Looking for accuracy improvement and overfitting prevention, given the imbalance in the classes composing the considered InSDN dataset (as detailed in Section 6.4.1), SMOTE has been applied to training data to generate synthetic minority-class samples.

More in detail, SMOTE addresses class imbalance by generating synthetic instances of the minority class rather than simply duplicating existing data points. This process involves selecting a sample from the minority class, finding its nearest neighbors in the feature space, and, then, creating new synthetic examples by interpolating between the chosen sample and its neighbors. This technique helps in enriching the dataset with more varied instances of the minority class, which, in turn, improves the performance of ML models by reducing their bias toward the majority class. The class distribution *before* and *after* the application of SMOTE on the considered InSDN dataset is detailed in Table 6.3 (last two columns).

6.4.2.4 Feature Selection

Then, in order to facilitate robust ML and DL experimental evaluations, flow-based features are extracted from the raw traffic which include over 80 features. This modern dataset addresses the shortcomings of legacy datasets by reflecting current attack vectors and the unique dynamics of SDN environments.

Moreover, in order to enhance model's efficiency and reduce its dimensionality, a feature selection using an RF classifier [182] has been performed. More in detail, RF is a supervised ensemble learning method constructing multiple Decision Trees (DTs) to evaluate feature importance on the basis of impurity reduction (also denoted as information gain). Then, by aggregating importance scores across trees, RF identifies the most influential features for prediction. Finally, these scores guide the selection of a subset of relevant features, reducing complexity and improving model performance. Thus, using this method, the ten most significant features shown in Figure 6.4 have been identified, as follows.

- `Dst Port`: corresponds to the destination port of the traffic flow.
- `Src Port`: corresponds to the source port of the traffic flow.
- `Init Bwd Win Byts`: corresponds to the initial bandwidth window (dimension: [bytes]) allocated for the specific traffic flow.
- `Bwd Header Len`: corresponds to the length of the header field reserved for the bandwidth indication.
- `Protocol`: corresponds to the bytes encoding representing the protocol adopted for transferring the specific traffic flow.
- `Fwd Header Len`: corresponds to the length (in bytes) of the headers in the forward direction of the flow.
- `Flow IAT Min`: indicates the shortest interval between packet arrivals in the flow.
- `Flow IAT Tot`: represents the total sum of inter-arrival times for packets within the flow.
- `Flow Duration`: corresponds to the time difference between the first and the last packet observed in the flow.
- `Fwd Pkts/s`: corresponds to the number of packets per second sent in the forward direction.

6.4.2.5 Feature Scaling

Finally, in order to ensure that all features equally contribute to the learning process, a standardization was applied to the considered InSDN dataset by exploiting the `StandardScaler` mechanism [183]. In this way, data are standardized by subtracting the mean and scaling to unit

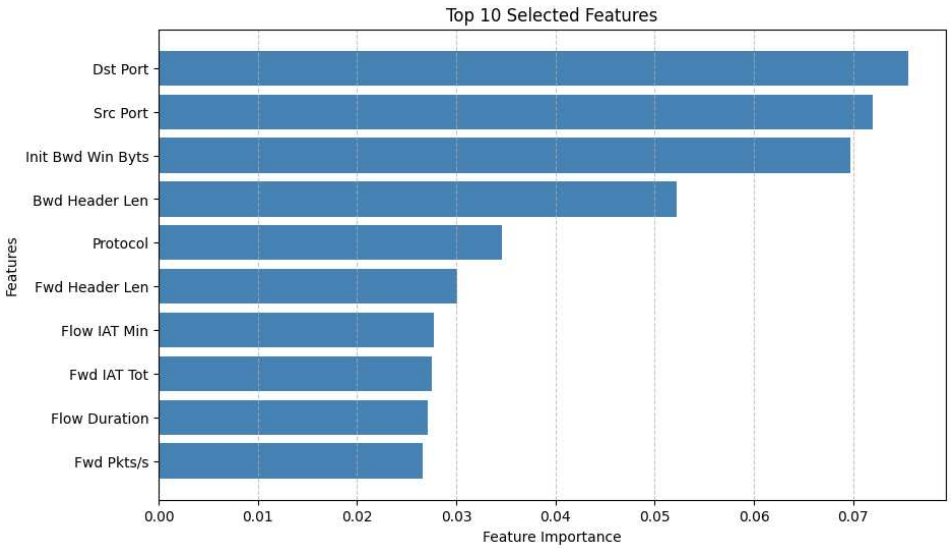


Figure 6.4: Dataset features importance score.

variance, placing all features on a comparable scale. On the analytical side, the resulting score (denoted as z) of a sample x is calculated as:

$$z = \frac{x - \mu}{\sigma} \tag{6.2}$$

where: μ and σ correspond to mean and variance of the training samples, respectively. This standardization ensures that features with different scales or units do not disproportionately affect the model, leading to better convergence during training and more interpretable results [184].

6.4.3 HyperParameter Optimization (HPO)

As anticipated in Section 6.1, in this research work different DL models are evaluated targeting a comprehensive assessment of their deployability on resource-constrained devices, at the same time providing high accuracy in detecting attacks in an SDN network environment. In order to achieve this, HPO [185] has been exploited to determine the optimal model configuration. More in detail, rather than maximizing accuracy alone, DL models were designed to minimize parameters count while sustaining high accuracy. As a result, by employing this optimization function, the optimal parameter set provides the lightest architecture with the best accuracy. Generally speaking, HPO involves the selection of the best hyperparameters for the specific model by minimizing an objective function that can be expressed as

$$\theta^* = \arg\min_{\theta \in \Theta} f(\theta) \tag{6.3}$$

where: θ represents the current set of hyperparameters; Θ corresponds to the hyperparameter search space; $f(\theta)$ is the objective function to be minimized.

To this regard, the specific objective function considered for HPO is the following:

$$f(\theta) = -\text{accuracy}(\theta) + k \cdot \text{complexity}(\theta) \tag{6.4}$$

where: accuracy(θ) measures the performance of the model (the higher its value, the better the performance); complexity(θ) quantifies the complexity of the model (e.g., the number of parameters or the computational costs); the term k is a constant controlling the trade-off between accuracy and complexity. Therefore, by minimizing $f(\theta)$, the optimization process inherently attempts to maximize accuracy while penalizing excessive complexity.

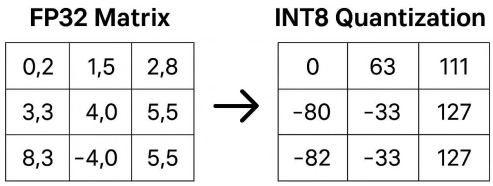


Figure 6.5: Representation of FP32 Weights to INT8 mapping process in PTQA.

6.4.4 Model Quantization

Quantization is a technique used to reduce the model size (denoted as $\mathcal{M}_{\text{size}}$) making it suitable for resource-constrained devices. Typically, DNNs are trained with FP32 numbers, which provide high precision at the cost of significant memory usage and computational power. Instead, quantization converts these high-precision numbers into formats with lower precision, such as INT8, thus reducing the models' memory footprints and speeding up calculations during inference. This process is particularly beneficial when using models on devices with limited resources, such as embedded systems.

6.4.4.1 Post-Training Quantization Aware (PTQA)

PTQA [186] optimizes pre-trained models by reducing computational requirements while maintaining accuracy. This is achieved by transforming model parameters (such as weights and activations), with inputs falling in the range $[-127, 127]$, enabling efficient use on microcontrollers that primarily support INT8 processing. More in detail, the INT8 quantization is performed as follows:

$$x_{\text{quantized}} = \text{round}\left(\frac{127}{\max(|x|)} \cdot x\right)$$

where: x represents the original floating-point values; $x_{\text{quantized}}$ represents the corresponding INT8 quantized values; $\max(|x|)$ represents the maximum value of x among all those considered; 127 is the maximum representable value in signed INT8 format.

For the sake of clarify, an example of FP32-to-INT8 weights quantization is shown in Figure 6.5.

Then, it is noteworthy to detail that PTQA can be categorized as follow.

- **Full quantization**: it converts *both* weights and activations to INT8 and ensures that all calculations use integer arithmetic. This maximizes performance and efficiency, but can lead to a higher loss of accuracy, especially for precision-sensitive models.
- **Dynamic quantization**: it converts *only* weights to INT8, while activations remain as FP32 or are dynamically quantized during inference. This approach preserves model accuracy better, but offers less speed and memory efficiency than full quantization.

6.4.5 Weight Pruning (WP)

A pruning schedule corresponds to a strategy or plan for gradually reducing the number of weights in a DL model, in detail making the model "sparser" by zeroing out weights over time. During the pruning schedule, less important weights are identified, based on the type of pruning [187]. Thus, pruning DL models produces a more compact network by eliminating elements such as neurons, filters, or channels (in the case of CNNs). This reduction in components leads to a smaller model with lower resource demands, but it can also decrease accuracy depending on the extent of pruning. For clarity, in Figure 6.6 a general WP implicitly removing neurons to lighten a DL model is shown.

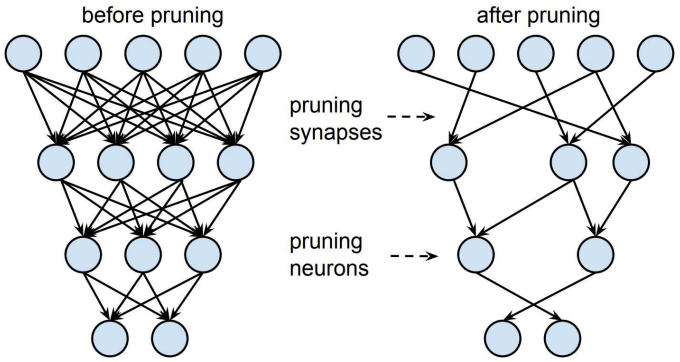


Figure 6.6: Exemplifier representation of the pruning process applied to a generic DL model.

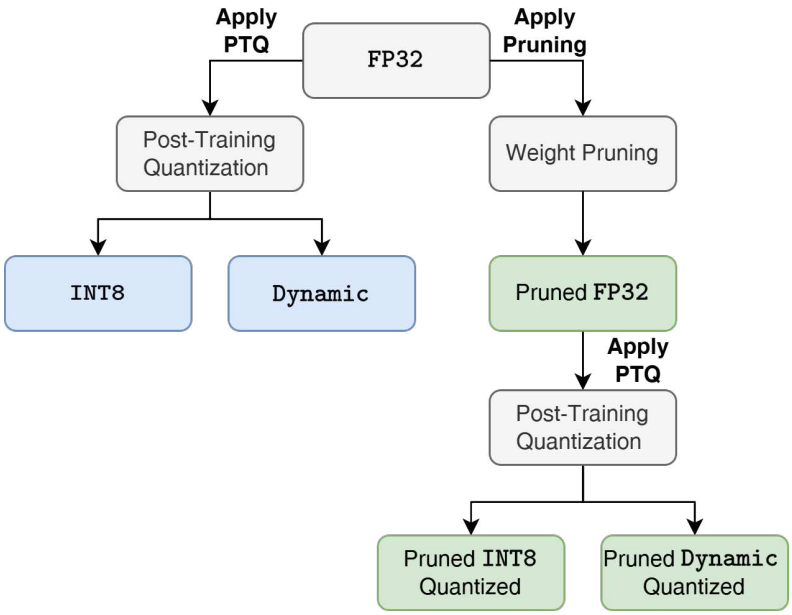


Figure 6.7: Proposed models PRIQ compression pipeline.

6.4.6 PRuning and Integer Quantization (PRIQ)

PRIQ aims to simultaneously leverage both pruning and quantization techniques in order to optimize model efficiency while maintaining accuracy. As a result, this approach takes advantage of both methodologies, optimizing the complexity reduction and retaining most of the precision.

For the sake of clarity, in Figure 6.7 the model compression pipeline starting from a `FP32` model and applying PTQ and WP, including combinations of both, is detailed.

6.4.7 Evaluation Metrics

The performance of the considered DL models has been assessed relying on the following well-known metrics. For the sake of clarity and completeness, it should be mentioned that recall (denoted as $\mathcal{R}$), a metric in general adopted for evaluating the performance of DL models and analytically corresponding to the ratio $\mathcal{R} = TP/TP+FN$, has not been considered as a performance metric because, for imbalanced datasets, the weighted recall equals the overall accuracy.

6.4.7.1 Accuracy

Accuracy (denoted as $\mathcal{A}$) measures the ratio between the number of correctly predicted instances and the total number of instances. It can be expressed as

$$\mathcal{A} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

where: TP, TN, FP, and FN represent the numbers of true positives, true negatives, false positives, and false negatives, respectively. The accuracy $\mathcal{A}$ ranges from 0 (with none of the instances being correctly classified) to 1 (with all instances being correctly classified). Although $\mathcal{A}$ is an indicator of the overall model performance, it may be misleading in the case of imbalanced datasets.

6.4.7.2 Precision

Precision (denoted as $\mathcal{P}$) indicates the ratio between the number of correctly predicted positive cases and the total number of positives. It can be expressed as

$$\mathcal{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}\,.$$

The precision ranges from 0 (worst case) to 1 (best case). Consequently, a high precision reduces the false positive rate, making it critical in applications where false positives are critical.

6.4.7.3 F1-Score

F1-Score (denoted as $\mathcal{F}$) balances precision $\mathcal{P}$ and recall $\mathcal{R}$ as follows:

$$\mathcal{F} = 2 \times \frac{\mathcal{P} \times \mathcal{R}}{\mathcal{P} + \mathcal{R}} = \frac{2 \times \mathrm{TP}}{2 \times \mathrm{TP} + \mathrm{FP} + \mathrm{FN}}\,.$$

The F1-Score ranges from 0 (if the precision $\mathcal{P}$ or the recall $\mathcal{R}$ is 0) to 1 (indicating perfect precision and recall). It is useful especially for imbalanced datasets, as it simultaneously takes into account both false positives and false negatives.

6.4.7.4 Cohen's Kappa Coefficient

The Cohen's Kappa Coefficient [188] (denoted as κ) is a statistical measure quantifying the agreement between *predicted* and *actual* labels, while adjusting for agreement occurring by chance. It is defined as follows:

$$\kappa = \frac{2 \times (\mathrm{TP} \times \mathrm{TN} - \mathrm{FN} \times \mathrm{FP})}{(\mathrm{TP} + \mathrm{FP}) \times (\mathrm{FP} + \mathrm{TN}) + (\mathrm{TP} + \mathrm{FN}) \times (\mathrm{FN} + \mathrm{TN})}\,.$$

Unlike accuracy, κ provides a more reliable evaluation in the presence of class imbalance. In fact, κ values range from -1 (complete disagreement) to $+1$ (perfect agreement), where 0 indicates an agreement on a random chance. In other words, the best case is with $|\kappa|$ close to 1.

6.4.7.5 Normalized Composite Accuracy-Size (NCAS) Score

In order to evaluate the trade-off between model size and accuracy drop after compression, we defined a specific score, named as Normalized Composite Accuracy-Size (NCAS) and denoted as $\mathcal{N}$, calculated for each considered DL model m as follows:

$$\mathcal{N}_m = \alpha \cdot (1 - \mathcal{D}_m) + (1 - \alpha) \cdot (1 - \mathcal{C}_m) \tag{6.5}$$

with:

$$\mathcal{C}_m = \frac{\widehat{\mathcal{M}_{\mathrm{size}_m}} - \min\left(\widehat{\mathcal{M}_{\mathrm{size}}}\right)}{\max\left(\widehat{\mathcal{M}_{\mathrm{size}}}\right) - \min\left(\widehat{\mathcal{M}_{\mathrm{size}}}\right)} \tag{6.6}$$

$$\mathcal{D}_m = \frac{\Delta\mathcal{A}_m - \min(\Delta\mathcal{A})}{\max(\Delta\mathcal{A}) - \min(\Delta\mathcal{A})} \tag{6.7}$$

$$\Delta\mathcal{A}_m = \mathcal{A}_m - \widehat{\mathcal{A}_m} \tag{6.8}$$

	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{F}$	κ
MLP	0.9968	0.9973	0.9970	0.9955
LSTM	**0.9979**	0.9980	**0.9979**	**0.9971**
GRU	0.9972	0.9973	0.9973	0.9962
BiLSTM	0.9964	0.9970	0.9966	0.9951
BiGRU	0.9934	0.9939	0.9935	0.9910
CNN	0.9914	0.9925	0.9917	0.9882
CNN-BiGRU	0.9944	0.9949	0.9946	0.9923
CNN-LSTM	0.9978	**0.9981**	**0.9979**	0.9970
CNN-GRU	0.9975	0.9977	0.9976	0.9966
CNN-BiLSTM	0.9974	0.9976	0.9975	0.9964

Table 6.4: Performance of the FP32 TF models according to the chosen evaluation metrics.

where:

- α corresponds to a weighting factor ($0 \leq \alpha \leq 1$) used to balance normalized accuracy drop (denoted as $\mathcal{D}_m$) *vs.* normalized compressed model size (denoted as $\mathcal{C}_m$); in this study, we consider $\alpha = 0.5$, thus assigning an equal relevance to both accuracy drop and compressed model size;
- $\widehat{\mathcal{M}_{\text{size}_m}}$ corresponds to the size (dimension: [KB]) of the m-th model after compression, applying quantization, pruning, or both techniques;
- $\min\left(\widehat{\mathcal{M}_{\text{size}}}\right)$ and $\max\left(\widehat{\mathcal{M}_{\text{size}}}\right)$ correspond to smallest and largest compressed model size among all models, respectively;
- $\widehat{\mathcal{A}_m}$ corresponds to the accuracy (dimension: [%]) of the m-th model after compression, applying quantization, pruning, or both techniques;
- $\Delta\mathcal{A}_m$ corresponds to the accuracy drop of the m-th model, as the difference between the accuracy of original and compressed versions of the same;
- $\min(\Delta\mathcal{A})$ and $\max(\Delta\mathcal{A})$ correspond to smallest and largest accuracy drop among all models, respectively.

Consequently, higher values of the term $(1 - \mathcal{D}_m)$ indicate better preservation of accuracy, while higher values of the term $(1 - \mathcal{C}_m)$ indicate smaller compressed model size.

6.5 Results and Discussion

6.5.1 Performance Evaluation

The performance of the considered DL models is evaluated according to the evaluation metrics detailed in Subsection 6.4.7, under the following four deployment configurations: (i) original FP32 TF models; (ii) pruned FP32 TF models; (iii) fully-quantized INT8 TFLite models; and (iv) PRIQ TFLite models.

6.5.1.1 Original FP32 TF Models

With regard to the evaluation of original FP32 TF models, the obtained results, shown in Table 6.4, show their ability to achieve high accuracy, with LSTM and CNN-LSTM overall performing best, each one exceeding 99.75% accuracy and 0.9976 F1-Score.

6.5.1.2 Pruned FP32 TF Models

The performance results of pruned FP32 TF models are shown in Table 6.5. It can be observed that the pruning operation degrades the performance minimally in most models. For instance,

	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{F}$	κ
MLP	0.9943	0.9953	0.9946	0.9922
LSTM	0.9969	0.9970	0.9969	0.9957
GRU	0.9970	0.9970	0.9970	0.9958
BiLSTM	0.9969	0.9972	0.9970	0.9958
BiGRU	0.9930	0.9935	0.9932	0.9904
CNN	0.9943	0.9948	0.9945	0.9922
CNN-BiGRU	0.9968	0.9970	0.9968	0.9956
CNN-LSTM	0.9981	0.9982	0.9981	0.9974
CNN-GRU	**0.9983**	**0.9984**	**0.9983**	**0.9977**
CNN-BiLSTM	0.9927	0.9930	0.9927	0.9899

Table 6.5: Performance of the pruned `FP32` TF models according to the chosen evaluation metrics.

	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{F}$	κ
MLP	0.9390	0.9505	0.9420	0.9184
LSTM	0.9031	0.9492	0.9231	0.8711
GRU	0.9259	0.9477	0.9331	0.8995
BiLSTM	0.9802	**0.9881**	0.9837	0.9731
BiGRU	0.9160	0.9411	0.9236	0.8892
CNN	0.9511	0.9632	0.9559	0.9334
CNN-BiGRU	0.9453	0.9562	0.9490	0.9255
CNN-LSTM	0.8885	0.9299	0.9033	0.8498
CNN-GRU	**0.9841**	0.9862	**0.9849**	**0.9782**
CNN-BiLSTM	0.5678	0.6536	0.4709	0.4338

Table 6.6: Performance of the `INT8` TFLite models according to the chosen evaluation metrics.

CNN-GRU slightly improves from its non-pruned counterpart in all metrics, reaching accuracy and F1-score equal to 99.83% and 0.9983, respectively.

We remark that hybrid CNN–RNN models maintain—or slightly improve—their post-pruning metrics. However, CNN-BiLSTM returns a limited post-pruning performance degradation. These findings suggest that pruning is a viable strategy to reduce the model size $\mathcal{M}_{\text{size}}$ without compromising performance, particularly for convolutional-recurrent hybrid models.

6.5.1.3 INT8 TFLite Models

With regard to the performance returned by the `INT8` TFLite models, quantizing to the `INT8` format yields variable effects: some models—such as CNN-GRU and BiLSTM—keep an accuracy higher than 98%, while others degrade sharply, as shown in Table 6.6. In particular, CNN-BiLSTM suffers from a severe degradation with an accuracy of 56.78% and an F1-score of 0.47, suggesting that quantization introduces numerical instability for certain architectures. Similarly, the performance of LSTM, CNN-LSTM, and MLP notably deteriorates under `INT8` quantization.

6.5.1.4 Pruned INT8 TFLite Models

According to the evaluation results detailed in Table 6.6 (regarding `INT8` TFLite models) and Table 6.7 (referring to their pruned versions), pruned `INT8` models tend to show better performance (in terms of accuracy $\mathcal{A}$, precision $\mathcal{P}$, F1-score $\mathcal{F}$, and Cohen's Kappa Coefficient κ) with respect to the original quantized `INT8` models, especially for the CNN-BiLSTM model, which returned significant improvements. However, for MLP and BiGRU, there was a performance degradation, according to several metrics, during pruning.

	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{F}$	κ
MLP	0.6921	0.8390	0.6847	0.5913
LSTM	0.9622	0.9667	0.9641	0.9482
GRU	0.8236	0.9020	0.8404	0.7640
BiLSTM	0.9682	0.9841	0.9755	0.9565
BiGRU	0.9023	0.9503	0.9220	0.8686
CNN	0.8545	0.9206	0.8583	0.8052
CNN-BiGRU	0.9366	0.9563	0.9442	0.9139
CNN-LSTM	0.9032	0.9243	0.9053	0.8689
CNN-GRU	**0.9861**	**0.9864**	**0.9862**	**0.9809**
CNN-BiLSTM	0.9706	0.9814	0.9751	0.9598

Table 6.7: Performance of the pruned INT8 TFLite models according to the chosen evaluation metrics.

	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{F}$	κ
MLP	**0.9962**	0.8789	0.8905	**0.9949**
LSTM	0.9864	0.8539	0.8621	0.9813
GRU	0.9907	0.9022	0.9236	0.9872
BiLSTM	0.9943	**0.9119**	**0.9424**	0.9922
BiGRU	0.9881	0.7982	0.8547	0.9837
CNN	0.9854	0.8618	0.8937	0.9800
CNN-BiGRU	0.9944	0.8706	0.9127	0.9923
CNN-LSTM	0.9920	0.8318	0.8788	0.9891
CNN-GRU	0.9938	0.8754	0.8971	0.9915
CNN-BiLSTM	0.6437	0.8825	0.7582	0.5270

Table 6.8: Performance of the dynamically-quantized TFLite models according to the chosen evaluation metrics.

	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{F}$	κ
MLP	0.9960	0.9966	0.9963	0.9946
LSTM	0.9862	0.9870	0.9862	0.9810
GRU	0.9909	0.9923	0.9913	0.9875
BiLSTM	0.9829	0.9891	0.9850	0.9766
BiGRU	0.9641	0.9896	0.9752	0.9512
CNN	0.9885	0.9891	0.9887	0.9843
CNN-BiGRU	**0.9968**	**0.9970**	**0.9968**	**0.9956**
CNN-LSTM	0.9878	0.9892	0.9882	0.9833
CNN-GRU	0.9924	0.9926	0.9924	0.9896
CNN-BiLSTM	0.9870	0.9926	0.9893	0.9822

Table 6.9: Performance of the pruned dynamically-quantized TFLite models according to the chosen evaluation metrics.

6.5.1.5 Normal and Pruned Dynamically-Quantized TFLite Models

Finally, on the basis of the results presented in Table 6.8 and Table 6.9, referring to normal and pruned dynamically-quantized TFLite models, respectively, it is evident that the combination of pruning and dynamical quantization generally enhances the performance, particularly in terms of precision and F1-Score. This effect is more pronounced in CNN-based architectures, where pruned models (such as CNN-GRU and CNN-BiGRU) achieve the best performance, with CNN-BiGRU returning the highest value in all metrics.

In fact, while dynamic quantization alone already provides significant memory efficiency, the addition of pruning seems to further refine model generalization by eliminating redundant or less influential weights. This reflects in more compact and potentially more robust models. The CNN-GRU model shows an improvement in F1-Score (from 0.9903 to 0.9924) and κ (from 0.9862 to

	Flash [KiB]	RAM [KiB]	No. Parameters	MACC	t_{exec} [ms]
MLP	17.78	2.22	2,016	2208	0.1584
LSTM	49.2	2.87	8,488	8648	0.9349
GRU	42.74	2.62	6,568	6528	0.6627
BiLSTM	84.57	3.73	16,968	17168	1.856
BiGRU	70.59	3.46	13,128	12928	1.302
CNN	27.64	2.53	3,784	4096	0.1888
CNN-BiGRU	39.55	3.35	4,344	4352	0.4678
CNN-LSTM	59.13	3.32	10,216	10400	0.8711
CNN-GRU	86.58	3.61	16,952	16928	1.180
CNN-BiLSTM	41.99	3.53	5,304	5472	0.6531

Table 6.10: Complexity evaluation of the considered FP32 TF models on a NUCLEO-G474RE resource-constrained device.

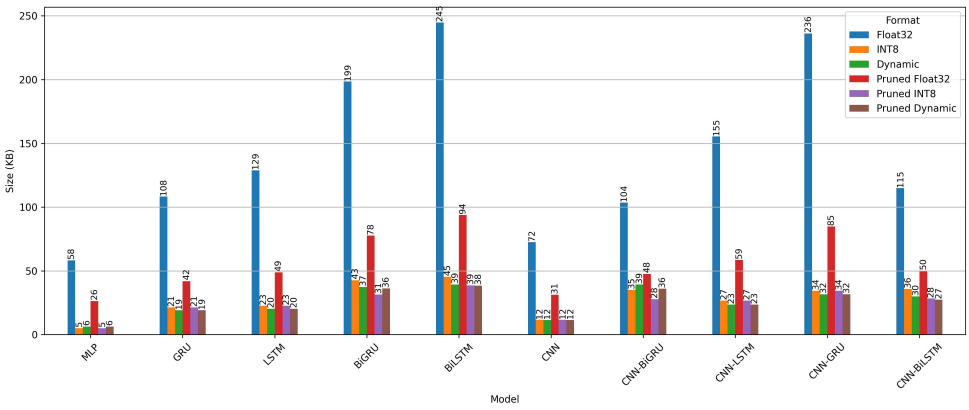


Figure 6.8: Comparison, in terms of memory footprints (dimension: [KiB]) for the considered DL models across different quantization and pruning techniques, with FP32 bars representing the original uncompressed models, while INT8, dynamic, and pruned bars corresponds to their variants.

0.9896) after pruning. Overall, the results confirm that pruning enhances the benefits of dynamic quantization, particularly for complex and hybrid models, such as CNN-RNN combinations.

6.5.2 Complexity Evaluation

Table 6.10 summarizes the computational complexity and resource usage of the various FP32 TF models evaluated on a resource-constrained embedded platform, namely NUCLEO-G474RE [189], featuring a 170 MHz Arm Cortex-M4 processor, 128 KB RAM, 512 KB internal flash storage. As evaluation metrics, flash size (dimension: [KiB]), RAM usage (dimension: [KiB]), number of parameters (adimensional), MACC operations (adimensional), and per-inference execution time (denoted as t_{exec}, dimension: [ms]) are considered.

Looking at the results shown in Table 6.10, among all models MLP exhibits the lowest complexity across all dimensions, only requiring 17.78 KiB of flash memory and 2.22 KiB of RAM, with the shortest execution time ($t_{exec} = 0.1584$ ms). This lightweight profile makes MLP particularly well-suited for deployment in resource-constrained devices such as the NUCLEO-G474RE board.

Then, for the sake of completeness, in Figure 6.8 the size reduction of the evaluated DL models using different compression techniques—including INT8 quantization, dynamic quantization, and pruning, applied both individually and in combination—is shown.

Among all models, MLP again achieves the highest reduction, with 91.35% size reduction using INT8 quantization (corresponding to an 11.57 compression ratio). Then, secondary INT8 optimal reduction ratios are returned by CNN-GRU and CNN models, with 85.47% and 84.01% reductions, respectively. Notably, dynamic quantization leads to an 86.60% reduction with

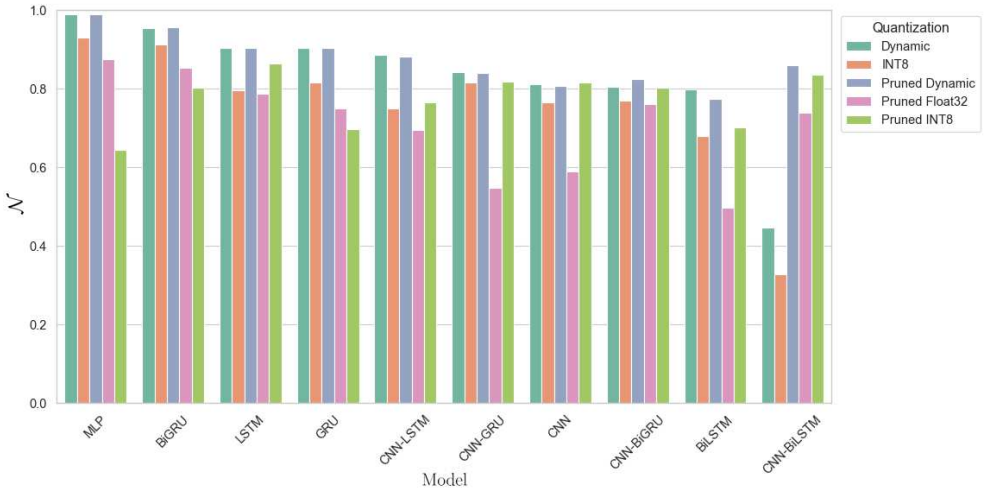


Figure 6.9: NCAS score $\mathcal{N}$ obtained across all the considered DL models and compression techniques.

Model	**Quantization Technique**	$\mathcal{M}_{\text{size}}$	$\widehat{\mathcal{M}_{\text{size}}}$	**Compression Ratio**	$\mathcal{A}$	$\widehat{\mathcal{A}}$	$\mathcal{N}$
MLP	Dynamic	58.29	6.35	9.18	0.9968	0.9962	0.98
BiGRU	Dynamic	72.48	11.53	6.29	0.9914	0.9854	0.95
LSTM	Dynamic	108.20	19.16	5.65	0.9979	0.9864	0.90
MLP	INT8	58.29	5.04	11.57	0.9968	0.9390	0.92
BiGRU	INT8	72.48	11.59	6.25	0.9914	0.9511	0.91
CNN-GRU	INT8	236.22	34.33	6.88	0.9975	0.9841	0.81
MLP	Pruned Dynamic	58.29	6.35	9.18	0.9968	0.9960	0.98
BiGRU	Pruned Dynamic	72.48	11.53	6.29	0.9914	0.9885	0.95
LSTM	Pruned Dynamic	108.20	19.16	5.65	0.9979	0.9862	0.90
MLP	Pruned FP32	58.29	26.32	2.21	0.9968	0.9943	0.87
BiGRU	Pruned FP32	72.48	31.17	2.33	0.9914	0.9943	0.85
LSTM	Pruned FP32	108.20	41.98	2.58	0.9979	0.9969	0.78
LSTM	Pruned INT8	108.20	21.32	5.08	0.9979	0.9622	0.86
CNN-BiLSTM	Pruned INT8	114.87	28.30	4.06	0.9974	0.9706	0.83
CNN-GRU	Pruned INT8	236.22	34.33	6.88	0.9975	0.9861	0.81

Table 6.11: Best three performing DL model per quantization technique.

CNN-GRU, which slightly outperforms its INT8 variant. Overall, pruning reduces FP32 sizes by 59.5%, while applying INT8 or dynamic quantization yields further reductions of 8.3% and 2.3%, respectively.

Finally, in Figure 6.9 we investigate the NCAS score $\mathcal{N}$ (detailed in Subsection 6.4.7.5) obtained across all the considered DL models and compression techniques—recall that higher scores indicate a more favorable balance between compactness and predictive performance.

Based on the results detailed in Table 6.11 and shown in Figure 6.9, MLP, BiGRU, and LSTM returned the highest $\mathcal{N}$ with dynamic quantization, as this technique provides substantial size reductions (over 80%) with negligible accuracy degradation (less than 1.2%). Notably, MLP (dynamic) model maintains an accuracy of 99.62% compared to its FP32 baseline (equal to

99.68%), while reducing the model size by nearly 90% and achieving an NCAS of 0.98.

With regard to INT8 quantization, the best-performing models (in terms of NCAS) are MLP, BiGRU, and CNN-GRU. Although INT8 quantization yields the largest models size reduction (up to 91.35%), it also introduces higher accuracy losses. In particular, MLP suffers a 5.78% accuracy drop, despite compressing the model size by about 91%.

Pruned dynamic quantization demonstrates performance comparable to those returned by dynamic quantization. For instance, BiGRU (pruned dynamic) exhibits a smaller accuracy drop (0.29%) compared to plain dynamic (0.60%), resulting in a slightly higher NCAS. Overall, as shown in Figure 6.9, values of $\mathcal{N}$ for pruned dynamic quantization are consistently high and often match or slightly surpass those obtained via standard dynamic quantization. This confirms that pruning combined with dynamic quantization preserves the balance between compression and accuracy.

In the case of pruned FP32 quantization, MLP achieves the highest NCAS score ($\mathcal{N} = 0.87$), with only a minor accuracy drop of 0.25% and a model size reduction of 54.8%. On average, pruned FP32 quantization reduces the model size by a 2.48x factor, while incurring an almost negligible accuracy drop of just 0.02%.

Finally, with regard to pruned INT8 quantization, best-performing models are LSTM, CNN-BiLSTM, and CNN-GRU, returning NCAS values of 0.86, 0.83, and 0.81, respectively. These models achieve model size reductions above 75% while maintaining accuracies above 97%. However, the accuracy losses (ranging from 1% to 3.6%) are more substantial compared to other pruning strategies, overall leading to relatively lower NCAS values. In this context, applying pruning together with INT8 quantization poses a particular challenge for MLP. Pruning removes information by zeroing out supposedly redundant weights, while quantization reduces the precision through value rounding. Unlike LSTM or GRU, MLP lacks memory mechanisms to compensate these losses, making it more prone to discarding subtle yet crucial details and, therefore, suffering a substantial drop in accuracy.

On the overall, dynamic quantization (both with and without pruning) consistently delivers excellent model size reduction with minimal accuracy drops (often less than 1%). Then, for several models (e.g., MLP, BiGRU, GRU, CNN-GRU), dynamic and pruned dynamic quantization yield similar and highly desirable outcomes. In fact, the NCAS scores for these techniques are generally high, highlighting a strong balance between models' compression and performance.

6.6 Conclusions

This work investigates the experimental performance evaluation of several DL models—namely: MLP, LSTM, GRU, BiLSTM, BiGRU, CNN, CNN-LSTM, CNN-GRU, CNN-BiLSTM, CNN-BiGRU—for intrusion detection in SDN-like scenarios. The performance is investigated, in a comparative way, according to well-known and widely-used evaluation metrics, including accuracy ($\mathcal{A}$), precision ($\mathcal{P}$), F1-Score ($\mathcal{F}$), and Cohen's Kappa Coefficient (κ).

Then, various model compression techniques—namely, PTQ, WP, and a combination of pruning and quantization (PRIQ)—have been applied in order to evaluate their impact on model size ($\mathcal{M}_{\text{size}}$) and prediction accuracy ($\mathcal{A}$). To this end, in order to systematically assess the balance between model's performance and efficiency, a trade-off score named as NCAS and quantifying the relationship between accuracy and model size, has been defined. On the overall, dynamic and pruned dynamic quantization emerged as the most effective techniques, by providing an average size reduction of 83.5% across all models while limiting the average accuracy drop to just 0.67%. Particularly, MLP and BiGRU offered a noticeable balance between accuracy and model size reduction in all of the cases except for pruned INT8 quantization, where LSTM and CNN-BiLSTM provide the highest NCAS but at the cost of significant accuracy degradation. These findings offer practical guidance for choosing and compressing DL models in resource-constrained SDN deployments. Future works will focus on extending the considered experimental framework to other domains, thus exploring adaptive compression strategies based on real-time resource availability.

Chapter 7

Conclusions

This thesis has addressed the challenge of developing and implementing intelligent, autonomous systems for monitoring and detecting anomalies in complex environments, ranging from the pharmaceutical industry to IoT and SDN networks. While the literature offers numerous AI solutions for these domains, they often remain theoretical due to their high computational cost or lack of adaptability. The research journey, articulated through five main studies, was driven by a dual objective: designing models capable of learning complex temporal dynamics while ensuring they are computationally efficient enough for real-world deployment on resource-constrained edge devices.

Summary of Findings

In Chapter 2 and 3, the research focused on fault detection for pharmaceutical processes. Unlike traditional monitoring methods found in the literature, which often rely on static thresholds and linear correlations, this work proposed a progression from PCA-based Health Indices to a more sophisticated predictive framework. By combining LSTM Autoencoders with a rolling-threshold mechanism, the system demonstrated a superior ability to handle non-stationary dynamics, providing more robust and adaptive detection compared to standard "one-size-fits-all" anomaly detection models.

The second phase addressed the critical gap between model performance and hardware deployability. In Chapter 4, the study on air quality prediction identified the hybrid CNN-BiGRU as an optimal balance. Crucially, while most current research focuses solely on accuracy, this study introduced a systematic application of PTQ techniques. This approach proved that it is possible to reduce model size by up to 66% with a minimal accuracy loss of approximately 1%, offering a pragmatic blueprint for implementing high-performance models on low-power sensors where standard deep learning architectures would typically fail.

In Chapter 5 and 6, the evaluation of DL for network intrusion detection led to a significant methodological contribution: the NCAS score. In contrast to existing literature that lacks standardized metrics for multi-objective evaluation (accuracy vs. complexity), the NCAS score provides a formal framework to justify the selection of lighter architectures like TCN and MLP over more complex variants. This demonstrates that for real-time security in IoT and SDN, optimized "lightweight" models are not just a compromise but a superior choice for operational stability and efficiency.

Limitations and Future Directions

Despite the promising results, this work acknowledges limitations that define future research paths. While the proposed autoencoder-based frameworks in Section 3.3.3 achieved reliable detection, their predictive capacity—often a weak point in current DL applications for industry—was only partially explored due to lack of data.

To address this, a preliminary integration of Koopman operators was attempted. This represents a significant shift from purely data-driven black-box models towards physics-informed AI, leveraging a mathematical framework to approximate nonlinear dynamics as linear in a higher-dimensional space. This allows the system to not only reconstruct current states but also proactively predict future behavior, potentially overcoming the latency issues inherent in reactive anomaly detection systems.

However, further investigation is required regarding the stability of Koopman embeddings and scalability to high-dimensional industrial data. Furthermore, while quantization and pruning were explored, other emerging methodologies like knowledge distillation or federated learning could further enhance privacy and efficiency in distributed environments.

Future directions include:

- Latent space analysis to evaluate how Koopman-based representations capture underlying system dynamics compared to standard latent mappings.
- Long-term validation of compressed models on physical hardware to measure actual energy consumption and latency under operational stress.
- The design of intrinsically lightweight neural architectures optimized specifically for the next generation of edge devices.
- The integration of online or incremental learning to enable models to evolve with new attack patterns or process drifts without full retraining.

In conclusion, this thesis demonstrates that the high complexity of modern DL models can be reconciled with the strict constraints of autonomous systems. By bridging the gap between theoretical accuracy and practical deployability, this research offers a methodological contribution that moves beyond the state-of-the-art, providing a viable path for the design of intelligent, efficient, and secure monitoring systems.

References

[1] Achouch, M. et al., "On predictive maintenance in industry 4.0: Overview, models, and challenges," *Applied Sciences*, vol. 12, no. 16, 2022. DOI: 10.3390/app12168081. [Online]. Available: https://www.mdpi.com/2076-3417/12/16/8081.

[2] Bousdekis, A., Lepenioti, K., Apostolou, D., and Mentzas, G., "A review of data-driven decision-making methods for industry 4.0 maintenance applications," *Electronics*, vol. 10, no. 7, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/7/828.

[3] Javaid, M., Haleem, A., Singh, R. P., and Suman, R., "Artificial intelligence applications for industry 4.0: A literature-based study," *Journal of Industrial Integration and Management*, vol. 07, no. 01, pp. 83–111, 2022. DOI: 10.1142/S2424862221300040. [Online]. Available: https://doi.org/10.1142/S2424862221300040.

[4] Javaid, M., Haleem, A., Singh, R. P., Suman, R., and Gonzalez, E. S., "Understanding the adoption of industry 4.0 technologies in improving environmental sustainability," *Sustainable Operations and Computers*, vol. 3, pp. 203–217, 2022. DOI: https://doi.org/10.1016/j.susoc.2022.01.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666412722000071.

[5] Keleko, A. T., Kamsu-Foguem, B., Ngouna, R. H., and Tongne, A., "Artificial intelligence and real-time predictive maintenance in industry 4.0: A bibliometric analysis," *AI and Ethics*, vol. 2, no. 4, pp. 553–577, Mar. 2022. DOI: 10.1007/s43681-021-00132-6. [Online]. Available: http://dx.doi.org/10.1007/s43681-021-00132-6.

[6] Kartheek, P., "Security and privacy technique in big data: A review," *North American Journal of Engineering Research*, vol. 5, no. 1, 2024.

[7] Hassoun, A. et al., "The fourth industrial revolution in the food industry—part i: Industry 4.0 technologies," *Critical Reviews in Food Science and Nutrition*, vol. 63, no. 23, pp. 6547–6563, 2023, PMID: 35114860. DOI: 10.1080/10408398.2022.2034735. [Online]. Available: https://doi.org/10.1080/10408398.2022.2034735.

[8] Abhinaya, N. et al., "A research on effective management of manufacturing defects to avoid product recalls: A challenge to pharmaceutical industry," *Research Journal of Pharmacy and Technology*, vol. 12, no. 12, p. 6124, 2019. DOI: 10.5958/0974-360x.2019.01064.3. [Online]. Available: http://dx.doi.org/10.5958/0974-360X.2019.01064.3.

[9] Mehta, A. K., Lanjewar, P., Murthy, D., Ghildiyal, P., Faldu, R., and L, N., "Ai & lean management principles based pharmaceutical manufacturing processes," in *2023 10th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, vol. 10, 2023, pp. 1599–1604. DOI: 10.1109/UPCON59197.2023.10434834.

[10] Fahmia, A. K., Kashyzadeh, K., and Ghorbania, S., "Enhanced autoregressive integrated moving average model for anomaly detection in power plant operations," *International Journal of Engineering*, vol. 37, no. 8, pp. 1691–1699, 2024.

[11] Guha, D., Chatterjee, R., and Sikdar, B., "Anomaly detection using lstm-based variational autoencoder in unsupervised data in power grid," *IEEE Systems Journal*, vol. 17, no. 3, pp. 4313–4323, 2023. DOI: 10.1109/JSYST.2023.3266554.

[12] Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., and Nikolopoulos, D. S., "Challenges and opportunities in edge computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, 2016, pp. 20–26. DOI: 10.1109/SmartCloud.2016.18.

[13] Shlezinger, N. and Bajić, I. V., "Collaborative inference for ai-empowered iot devices," *IEEE Internet of Things Magazine*, vol. 5, no. 4, pp. 92–98, 2022. DOI: 10.1109/IOTM.001.2200152.

[14] P, A., Chouhan, N., Chandhok, G. A., Sugumaran, D., Aswal, U., and A, S., "Empowering iot devices with energy-efficient ai and machine learning," in *2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT)*, vol. 1, 2024, pp. 720–725. DOI: 10.1109/ICCPCT61902.2024.10672916.

[15] Liang, A., Hu, Y., and Li, G., "The impact of improved pca method based on anomaly detection on chiller sensor fault detection," *International journal of refrigeration*, vol. 155, pp. 184–194, 2023.

[16] Noh, S.-H., "Analysis of gradient vanishing of rnns and performance comparison," *Information*, vol. 12, no. 11, 2021. DOI: 10.3390/info12110442. [Online]. Available: https://www.mdpi.com/2078-2489/12/11/442.

[17] Noh, S.-H., "Analysis of gradient vanishing of rnns and performance comparison," *Information*, vol. 12, no. 11, p. 442, 2021.

[18] Fan, J., Zhang, K., Huang, Y., Zhu, Y., and Chen, B., "Parallel spatio-temporal attention-based tcn for multivariate time series prediction," *Neural Computing and Applications*, vol. 35, no. 18, pp. 13 109–13 118, May 2021. DOI: 10.1007/s00521-021-05958-z. [Online]. Available: http://dx.doi.org/10.1007/s00521-021-05958-z.

[19] Mazinani, A., Antonucci, D., Pietro Pau, D., Davoli, L., and Ferrari, G., "Air quality prediction via embedded ml/dl and quantized models," *IEEE Access*, vol. 13, pp. 154 203–154 218, 2025. DOI: 10.1109/ACCESS.2025.3603920.

[20] Nail, S. L., Jiang, S., Chongprasert, S., and Knopp, S. A., "Fundamentals of freeze-drying," in *Development and Manufacture of Protein Pharmaceuticals*. Springer US, 2002, pp. 281–360. DOI: 10.1007/978-1-4615-0549-5_6. [Online]. Available: http://dx.doi.org/10.1007/978-1-4615-0549-5_6.

[21] Sahni, E. et al., "Lyophilizer leak rate testing – an industry survey and best practice recommendation," *Journal of Pharmaceutical Sciences*, vol. 111, no. 10, pp. 2714–2718, Oct. 2022. DOI: 10.1016/j.xphs.2022.06.025. [Online]. Available: http://dx.doi.org/10.1016/j.xphs.2022.06.025.

[22] Oliosi, E., Calzavara, G., and Ferrari, G., "On sensor data clustering for machine status monitoring and its application to predictive maintenance," *IEEE Sensors Journal*, vol. 23, no. 9, pp. 9620–9639, May 2023. DOI: 10.1109/jsen.2023.3260314. [Online]. Available: http://dx.doi.org/10.1109/JSEN.2023.3260314.

[23] Calzavara, G., Consolini, L., and Ferrari, G., "Leak detection and classification in pharmaceutical freeze-dryers: An identification-based approach," in *2021 60th IEEE Conference on Decision and Control (CDC)*, IEEE, Dec. 2021. DOI: 10.1109/cdc45484.2021.9683753. [Online]. Available: http://dx.doi.org/10.1109/CDC45484.2021.9683753.

[24] Koganti, V. R. et al., "Investigation of design space for freeze-drying: Use of modeling for primary drying segment of a freeze-drying cycle," *AAPS PharmSciTech*, vol. 12, no. 3, pp. 854–861, Jun. 2011. DOI: 10.1208/s12249-011-9645-7. [Online]. Available: http://dx.doi.org/10.1208/s12249-011-9645-7.

[25] Velardi, S. A. and Barresi, A. A., "Development of simplified models for the freeze-drying process and investigation of the optimal operating conditions," *Chemical Engineering Research and Design*, vol. 86, no. 1, pp. 9–22, Jan. 2008. DOI: 10.1016/j.cherd.2007.10.007. [Online]. Available: http://dx.doi.org/10.1016/j.cherd.2007.10.007.

[26] Maltesen, M. J., Bjerregaard, S., Hovgaard, L., Havelund, S., Weert, M. van de, and Grohganz, H., "Multivariate analysis of phenol in freeze-dried and spray-dried insulin formulations by nir and ftir," *AAPS PharmSciTech*, vol. 12, no. 2, pp. 627–636, May 2011. DOI: 10.1208/s12249-011-9618-x. [Online]. Available: http://dx.doi.org/10.1208/s12249-011-9618-x.

[27] Bro, R. and Smilde, A. K., "Principal component analysis," *Anal. Methods*, vol. 6, no. 9, pp. 2812–2831, 2014. DOI: 10.1039/c3ay41907j. [Online]. Available: http://dx.doi.org/10.1039/C3AY41907J.

[28] Majozi, T., "Introduction to batch chemical processes," *Batch Chemical Process Integration: Analysis, Synthesis and Optimization*, pp. 1–11, Nov. 2009. [Online]. Available: https://doi.org/10.1007/978-90-481-2588-3.

[29] Mockus, L., Peterson, J. J., Lainez, J. M., and Reklaitis, G. V., "Batch-to-batch variation: A key component for modeling chemical manufacturing processes," *Organic Process Research & Development*, vol. 19, no. 8, pp. 908–914, Oct. 2015. [Online]. Available: https://doi.org/10.1021/op500244f.

[30] Greenacre, M., Groenen, P. J., Hastie, T., d'Enza, A. I., Markos, A., and Tuzhilina, E., "Principal component analysis," *Nature Reviews Methods Primers*, vol. 2, no. 1, p. 100, Dec. 2022. [Online]. Available: https://doi.org/10.1038/s43586-022-00184-w.

[31] Pirouz, D. M., "An overview of partial least squares," *Available at SSRN 1631359*, Jun. 2006. [Online]. Available: https://dx.doi.org/10.2139/ssrn.1631359.

[32] Schölkopf, B., Smola, A., and Müller, K.-R., "Kernel principal component analysis," in *International conference on artificial neural networks*, Springer, Jun. 1997, pp. 583–588. [Online]. Available: https://doi.org/10.1007/BFb0020217.

[33] Aghaee, M., Mishra, A., Krau, S., Tamer, I. M., and Budman, H., "Artificial intelligence applications for fault detection and diagnosis in pharmaceutical bioprocesses: A review," *Current Opinion in Chemical Engineering*, vol. 44, p. 101 025, Jun. 2024. [Online]. Available: https://doi.org/10.1016/j.coche.2024.101025.

[34] Russell, E. L., Chiang, L. H., and Braatz, R. D., "Fault detection in industrial processes using canonical variate analysis and dynamic principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 51, no. 1, pp. 81–93, May 2000. [Online]. Available: https://doi.org/10.1016/S0169-7439(00)00058-7.

[35] Jeffy, F., Gugaliya, J. K., and Kariwala, V., "Application of multi-way principal component analysis on batch data," in *2018 UKACC 12th International Conference on Control (CONTROL)*, IEEE, 2018, pp. 414–419. [Online]. Available: https://doi.org/10.1109/CONTROL.2018.8516806.

[36] Choi, S. W., Lee, C., Lee, J.-M., Park, J. H., and Lee, I.-B., "Fault detection and identification of nonlinear processes based on kernel pca," *Chemometrics and intelligent laboratory systems*, vol. 75, no. 1, pp. 55–67, Jan. 2005. [Online]. Available: https://doi.org/10.1016/j.chemolab.2004.05.001.

[37] Li, K.-L., Huang, H.-K., Tian, S.-F., and Xu, W., "Improving one-class svm for anomaly detection," in *Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE Cat. No. 03EX693)*, IEEE, vol. 5, 2003, pp. 3077–3081. [Online]. Available: https://doi.org/10.1109/ICMLC.2003.1260106.

[38] Zhao, Y., Wang, S., and Xiao, F., "Pattern recognition-based chillers fault detection method using support vector data description (svdd)," *Applied Energy*, vol. 112, pp. 1041–1048, Dec. 2013. [Online]. Available: https://doi.org/10.1016/j.apenergy.2012.12.043.

[39] Inoue, J., Yamagata, Y., Chen, Y., Poskitt, C. M., and Sun, J., "Anomaly detection for a water treatment system using unsupervised machine learning," in *2017 IEEE international conference on data mining workshops (ICDMW)*, IEEE, Nov. 2017, pp. 1058–1065. [Online]. Available: https://doi.org/10.1109/ICDMW.2017.149.

[40] Kilickaya, S. et al., "Audio-based anomaly detection in industrial machines using deep one-class support vector data description," *arXiv preprint arXiv:2412.10792*, Dec. 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.10792.

[41] Sakurada, M. and Yairi, T., "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, Dec. 2014, pp. 4–11. [Online]. Available: https://doi.org/10.1145/2689746.2689747.

[42] Said Elsayed, M., Le-Khac, N.-A., Dev, S., and Jurcut, A. D., "Network anomaly detection using lstm based autoencoder," in *Proceedings of the 16th ACM symposium on QoS and security for wireless and mobile networks*, Nov. 2020, pp. 37–45. [Online]. Available: https://doi.org/10.1145/3416013.3426457.

[43] Nguyen, H. D., Tran, K. P., Thomassey, S., and Hamad, M., "Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management," *International Journal of Information Management*, vol. 57, p. 102 282, Apr. 2021. [Online]. Available: https://doi.org/10.1016/j.ijinfomgt.2020.102282.

[44] Liu, M., Zhu, T., Ye, J., Meng, Q., Sun, L., and Du, B., "Spatio-temporal autoencoder for traffic flow prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 5, pp. 5516–5526, Feb. 2023. [Online]. Available: https://doi.org/10.1109/TITS.2023.3243913.

[45] Zeng, L., Long, W., and Li, Y., "A novel method for gas turbine condition monitoring based on kpca and analysis of statistics t2 and spe," *Processes*, vol. 7, no. 3, p. 124, Feb. 2019. [Online]. Available: https://doi.org/10.3390/pr7030124.

[46] Lee, J.-M., Yoo, C., and Lee, I.-B., "Fault detection of batch processes using multiway kernel principal component analysis," *Computers & chemical engineering*, vol. 28, no. 9, pp. 1837–1847, Aug. 2004. [Online]. Available: https://doi.org/10.1016/j.compchemeng.2004.02.036.

[47] Ahsan, M. M., Mahmud, M. P., Saha, P. K., Gupta, K. D., and Siddique, Z., "Effect of data scaling methods on machine learning algorithms and model performance," *Technologies*, vol. 9, no. 3, p. 52, Jul. 2021. [Online]. Available: https://doi.org/10.3390/technologies9030052.

[48] Torres, J. F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., and Troncoso, A., "Deep learning for time series forecasting: A survey," *Big data*, vol. 9, no. 1, pp. 3–21, 2021. [Online]. Available: https://doi.org/10.1089/big.2020.0159.

[49] Kong, Y. et al., "Unlocking the power of lstm for long term time series forecasting," *arXiv preprint arXiv:2408.10006*, Apr. 2024. [Online]. Available: https://doi.org/10.1609/aaai.v39i11.33303.

[50] Wu, J. et al., "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019. [Online]. Available: 10.11989/JEST.1674-862X.80904120.

[51] Sabzehi, M. and Rollins, P., "Enhancing rover mobility monitoring: Autoencoder-driven anomaly detection for curiosity," in *2024 IEEE Aerospace Conference*, IEEE, 2024, pp. 1–7. [Online]. Available: https://doi.org/10.1109/AERO58975.2024.10521179.

[52] Ghorbani, H., "Mahalanobis distance and its application for detecting multivariate outliers," *Facta Universitatis, Series: Mathematics and Informatics*, pp. 583–595, 2019. [Online]. Available: https://doi.org/10.22190/FUMI1903583G.

[53] Han, S., Hu, X., Huang, H., Jiang, M., and Zhao, Y., "Adbench: Anomaly detection benchmark," *Advances in neural information processing systems*, vol. 35, pp. 32 142–32 159, 2022.

[54] Dominski, F. H. et al., "Effects of Air Pollution on Health: A Mapping Review of Systematic Reviews and Meta-Analyses," *Environmental Research*, vol. 201, p. 111 487, 2021, doi:10.1016/j.envres.2021.111487. DOI: 10.1016/j.envres.2021.111487.

[55] Hamanaka, R. B. and Mutlu, G. M., "Particulate matter air pollution: Effects on the cardiovascular system," *Frontiers in Endocrinology*, vol. 9, Nov. 2018, doi:10.3389/fendo.2018.00680. DOI: 10.3389/fendo.2018.00680.

[56] BBae, S. and Hong, Y.-C., "Health effects of particulate matter," *Journal of the Korean Medical Association*, vol. 61, no. 12, pp. 749–755, 2018, doi:10.5124/jkma.2018.61.12.749. DOI: 10.5124/jkma.2018.61.12.749.

[57] Arias-Pérez, R. D. et al., "Inflammatory effects of particulate matter air pollution," *Environmental Science and Pollution Research*, vol. 27, no. 34, pp. 42 390–42 404, 2020, doi:10.1007/s11356-020-10574-w. DOI: 10.1007/s11356-020-10574-w.

[58] Fu, P. and Yung, K. K. L., "Air pollution and alzheimer's disease: A systematic review and meta-analysis," *Journal of Alzheimer's disease*, vol. 77, no. 2, pp. 701–714, 2020.

[59] Murata, H., Barnhill, L. M., and Bronstein, J. M., "Air pollution and the risk of parkinson's disease: A review," *Movement Disorders*, vol. 37, no. 5, pp. 894–904, 2022.

[60] Stevens, C. J. et al., "The impact of air pollution on terrestrial managed and natural vegetation," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2183, p. 20 190 317, Sep. 2020, doi:10.1098/rsta.2019.0317. DOI: 10.1098/rsta.2019.0317.

[61] Sofia, D. et al., "Mitigation strategies for reducing air pollution," *Environmental Science and Pollution Research*, vol. 27, no. 16, pp. 19 226–19 235, 2020, doi:10.1007/s11356-020-08647-x. DOI: 10.1007/s11356-020-08647-x.

[62] Kwilinski, A., Lyulyov, O., and Pimonenko, T., "Reducing transport sector CO2 emissions patterns: Environmental technologies and renewable energy," *Journal of Open Innovation: Technology, Market, and Complexity*, vol. 10, no. 1, p. 100 217, 2024, doi:10.1016/j.joitmc.2024.100217. DOI: 10.1016/j.joitmc.2024.100217.

[63] Somvanshi, S. S. et al., "Delhi Air Pollution Modeling Using Remote Sensing Technique," in *Handbook of Environmental Materials Management*, Hussain, C. M., Ed. Springer International Publishing, 2019, pp. 1–27, doi:10.1007/978-3-319-58538-3_174-1. DOI: 10.1007/978-3-319-58538-3_174-1.

[64] Chen, Y. et al., "Multifactor Spatio-Temporal Correlation Model Based on a Combination of Convolutional Neural Network and Long Short-Term Memory Neural Network for Wind Speed Forecasting," *Energy Conversion and Management*, vol. 185, pp. 783–799, 2019, doi:10.1016/j.enconman.2019.02.018. DOI: 10.1016/j.enconman.2019.02.018.

[65] Ma, J. et al., "Improving Air Quality Prediction Accuracy at Larger Temporal Resolutions using Deep Learning and Transfer Learning Techniques," *Atmospheric Environment*, vol. 214, p. 116 885, 2019, doi:10.1016/j.atmosenv.2019.116885. DOI: 10.1016/j.atmosenv.2019.116885.

[66] Al-Eidi, S. et al., "Comparative Analysis Study for Air Quality Prediction in Smart Cities Using Regression Techniques," *IEEE Access*, vol. 11, pp. 115 140–115 149, 2023, doi:10.1109/ACCESS.2023.3323447. DOI: 10.1109/ACCESS.2023.3323447.

[67] Kaur, M. et al., "Computational deep air quality prediction techniques: A systematic review," *Artificial Intelligence Review*, vol. 56, no. Suppl 2, pp. 2053–2098, 2023, doi:10.1007/s10462-023-10570-9. DOI: 10.1007/s10462-023-10570-9.

[68] Hua, Y. et al., "Deep Learning with Long Short-Term Memory for Time Series Prediction," *IEEE Communications Magazine*, vol. 57, no. 6, pp. 114–119, 2019, doi:10.1109/MCOM.2019.1800155. DOI: 10.1109/MCOM.2019.1800155.

[69] Zhai, N., Yao, P., and Zhou, X., "Multivariate Time Series Forecast in Industrial Process Based on XGBoost and GRU," in *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, doi:10.1109/ITAIC49862.2020.9338878, vol. 9, Chongqing, China, 2020, pp. 1397–1400. DOI: 10.1109/ITAIC49862.2020.9338878.

[70] Mohammadi, F. et al., "Prediction of atmospheric PM2. 5 level by machine learning techniques in Isfahan, Iran," *Scientific Reports*, vol. 14, no. 2109, 2024, doi:10.1038/s41598-024-52617-z. DOI: 10.1038/s41598-024-52617-z.

[71] Thaweephol, K. and Wiwatwattana, N., "Long Short-Term Memory Deep Neural Network Model for PM2.5 Forecasting in the Bangkok Urban Area," in *2019 17th International Conference on ICT and Knowledge Engineering (ICT&KE)*, doi:10.1109/ICTKE47035.2019.8966854, Bangkok, Thailand, 2019, pp. 1–6. DOI: 10.1109/ICTKE47035.2019.8966854.

[72] Alharbi, F. R. and Csala, D., "A Seasonal Autoregressive Integrated Moving Average with Exogenous Factors (SARIMAX) Forecasting Model-Based Time Series Approach," *Inventions*, vol. 7, no. 4, 2022, doi:10.3390/inventions7040094. DOI: 10.3390/inventions7040094.

[73] Chang, Y.-S. et al., "An LSTM-based aggregated model for air pollution forecasting," *Atmospheric Pollution Research*, vol. 11, no. 8, pp. 1451–1463, 2020, doi:10.1016/j.apr.2020.05.015. DOI: 10.1016/j.apr.2020.05.015.

[74] Usha Ruby, A. et al., "Forecasting PM2.5 Concentration Using Gradient-Boosted Regression Tree with CNN Learning Model," *Optical Memory and Neural Networks*, vol. 33, no. 1, pp. 86–96, 2024, doi:10.3103/s1060992x24010107. DOI: 10.3103/s1060992x24010107.

[75] Luo, A. et al., "Application of accurate online support vector regression in atmospheric SO2 concentration prediction," in *2018 Chinese Control And Decision Conference (CCDC)*, doi:10.1109/CCDC.2018.8408231, Shenyang, China, 2018, pp. 6274–6279. DOI: 10.1109/CCDC.2018.8408231.

[76] Choi, S. W. and Kim, B. H. S., "Applying PCA to Deep Learning Forecasting Models for Predicting PM2.5," *Sustainability*, vol. 13, no. 7, 2021, doi:10.3390/su13073726. DOI: 10.3390/su13073726.

[77] Greenacre, M. et al., "Principal component analysis," *Nature Reviews Methods Primers*, vol. 2, no. 1, p. 100, 2022, doi:10.1038/s43586-022-00184-w. DOI: 10.1038/s43586-022-00184-w.

[78] Ding, C. et al., "A hybrid CNN-LSTM model for predicting PM2.5 in Beijing based on spatiotemporal correlation," *Environmental and Ecological Statistics*, 2021, doi:10.1007/s10651-021-00501-8. DOI: 10.1007/s10651-021-00501-8.

[79] Tao, Q. et al., "Air Pollution Forecasting Using a Deep Learning Model Based on 1D Convnets and Bidirectional GRU," *IEEE Access*, vol. 7, pp. 76 690–76 698, 2019, doi:10.1109/ACCESS.2019.2921578. DOI: 10.1109/ACCESS.2019.2921578.

[80] Kim, Y.-b. et al., "Comparison of PM2.5 prediction performance of the three deep learning models: A case study of Seoul, Daejeon, and Busan," *Journal of Industrial and Engineering Chemistry*, vol. 120, pp. 159–169, 2023, doi:10.1016/j.jiec.2022.12.022. DOI: 10.1016/j.jiec.2022.12.022.

[81] Eren, B., Aksangür, İ., and Erden, C., "Predicting next hour fine particulate matter (PM2.5) in the Istanbul Metropolitan City using deep learning algorithms with time windowing strategy," *Urban Climate*, vol. 48, p. 101 418, 2023, doi:10.1016/j.uclim.2023.101418. DOI: 10.1016/j.uclim.2023.101418.

[82] Zhu, M. and Xie, J., "Investigation of nearby monitoring station for hourly PM2.5 forecasting using parallel multi-input 1D-CNN-biLSTM," *Expert Systems with Applications*, vol. 211, p. 118 707, 2023, doi:10.1016/j.eswa.2022.118707. DOI: 10.1016/j.eswa.2022.118707.

[83] Datta, A. et al., "Efficient Air Quality Index Prediction on Resource-Constrained Devices using TinyML: Design, Implementation, and Evaluation," in *25th International Conference on Distributed Computing and Networking*, doi:10.1145/3631461.3631956, Chennai, India, 2024, pp. 304–309. DOI: 10.1145/3631461.3631956.

[84] Mazinani, A. et al., "Air Quality Estimation with Embedded AI-Based Prediction Algorithms," in *2023 International Conference on Information Technology Research and Innovation (ICITRI)*, doi:10.1109/ICITRI59340.2023.10249864, Jakarta, Indonesia, 2023, pp. 87–92. DOI: 10.1109/ICITRI59340.2023.10249864.

[85] Wardana, I. N. K., Fahmy, S. A., and Gardner, J. W., "TinyML Models for a Low-Cost Air Quality Monitoring Device," *IEEE Sensors Letters*, vol. 7, no. 11, pp. 1–4, 2023, doi:10.1109/LSENS.2023.3315249. DOI: 10.1109/LSENS.2023.3315249.

[86] scikit-learn, *Imputation of missing values*, https://scikit-learn.org/stable/modules/impute.html. Accessed on July 28, 2025.

[87] Voelker, A., Kajić, I., and Eliasmith, C., "Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 32, 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/952285b9b7e7a1be5aa7849f32ffff05-Paper.pdf.

[88] Bansal, M., Goyal, A., and Choudhary, A., "A comparative analysis of k-nearest neighbor, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning," *Decision Analytics Journal*, vol. 3, p. 100 071, 2022.

[89] Mienye, I. D. and Sun, Y., "A survey of ensemble learning: Concepts, algorithms, applications, and prospects," *Ieee Access*, vol. 10, pp. 99 129–99 149, 2022.

[90] Mazinani, A., Pau, D. P., Davoli, L., and Ferrari, G., "Deep Neural Quantization for Speech Detection of Parkinson Disease," in *2024 IEEE 8th Forum on Research and Technologies for Society and Industry Innovation (RTSI)*, doi:10.1109/RTSI61910.2024.10761283, Milan, Italy, Sep. 2024, pp. 178–183. DOI: 10.1109/RTSI61910.2024.10761283.

[91] Li, G. et al., "Pearson Correlation Coefficient-Based Performance Enhancement of Broad Learning System for Stock Price Prediction," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 5, pp. 2413–2417, 2022, doi: 10.1109/TCSII.2022.3160266. DOI: 10.1109/TCSII.2022.3160266.

[92] scikit-learn, *MinMaxScaler*, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. Accessed on August 26, 2025.

[93] Wu, J. et al., "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019, doi: 10.11989/JEST.1674-862X.80904120. DOI: 10.11989/JEST.1674-862X.80904120.

[94] Yang, L. and Shami, A., "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020, doi: 10.1016/j.neucom.2020.07.061. DOI: 10.1016/j.neucom.2020.07.061.

[95] Gramacy, R. B., *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. Chapman and Hall/CRC, Mar. 2020. DOI: 10.1201/9780367815493. [Online]. Available: https://doi.org/10.1201/9780367815493.

[96] Pau, D., Pisani, A., and Candelieri, A., "Towards Full Forward On-Tiny-Device Learning: A Guided Search for a Randomly Initialized Neural Network," *Algorithms*, vol. 17, no. 1, 2024, doi: 10.3390/a17010022. DOI: 10.3390/a17010022.

[97] Hoof, J. van and Vanschoren, J., "Hyperboost: Hyperparameter optimization by gradient boosting surrogate models," *arXiv preprint arXiv:2101.02289*, 2021, doi: 10.48550/arXiv.2101.02289. DOI: 10.48550/arXiv.2101.02289.

[98] Bischl, B. et al., "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, no. 2, e1484, 2023, doi: 10.1002/widm.1484. DOI: 10.1002/widm.1484.

[99] Lindauer, M. et al., "SMAC3: A versatile Bayesian optimization package for hyperparameter optimization," *Journal of Machine Learning Research*, vol. 23, no. 54, pp. 1–9, 2022.

[100] Davoli, L., Belli, L., and Ferrari, G., *Indoor Air Quality Monitoring @ Brindisi Airport*, doi: 10.17632/BV2HVM4PMZ, 2023. DOI: 10.17632/BV2HVM4PMZ.

[101] Davoli, L., Belli, L., and Ferrari, G., "Air quality dataset from an indoor airport travelers transit area," *Data in Brief*, vol. 52, p. 109 821, 2024, doi: 10.1016/j.dib.2023.109821. DOI: 10.1016/j.dib.2023.109821.

[102] STMicroelectronics, *Discovery kit with STM32F469NI MCU*, https://www.st.com/en/evaluation-tools/32f469idiscovery.html. Accessed on August 26, 2025.

[103] STMicroelectronics, *ST Edge AI Developer Cloud*, https://stm32ai.st.com/st-edge-ai-developer-cloud/. Accessed on August 26, 2025.

[104] STMicroelectronics, *STM32CubeMX*, https://www.st.com/en/development-tools/stm32cubemx.html. Accessed on August 26, 2025, 2024.

[105] Google AI Edge, *LiteRT – Post-training quantization*, https://ai.google.dev/edge/litert/models/post_training_quantization. Accessed on August 26, 2025.

[106] Rejeb, A. et al., "The Internet of Things and the Circular Economy: A Systematic Literature Review and Research Agenda," *Journal of Cleaner Production*, vol. 350, p. 131 439, 2022, doi:10.1016/j.jclepro.2022.131439. DOI: 10.1016/j.jclepro.2022.131439.

[107] Premalatha, B. and Prakasam, P., "A Review on FoG Computing in 5G Wireless Technologies: Research Challenges, Issues and Solutions," *Wireless Personal Communications*, vol. 134, no. 4, pp. 2455–2484, Feb. 2024, doi:10.1007/s11277-024-11061-y. DOI: 10.1007/s11277-024-11061-y.

[108] Zikria, Y. B. et al., "Next-Generation Internet of Things (IoT): Opportunities, Challenges, and Solutions," *Sensors*, vol. 21, no. 4, p. 1174, Feb. 2021, doi:10.3390/s21041174. DOI: 10.3390/s21041174.

[109] Heidari, A. and Jabraeil Jamali, M. A., "Internet of Things Intrusion Detection Systems: A Comprehensive Review and Future Directions," *Cluster Computing*, vol. 26, no. 6, pp. 3753–3780, Oct. 2022, doi:10.1007/s10586-022-03776-z. DOI: 10.1007/s10586-022-03776-z.

[110] Jiang, X., Lora, M., and Chattopadhyay, S., "An Experimental Analysis of Security Vulnerabilities in Industrial IoT Devices," *ACM Transactions on Internet Technology*, vol. 20, no. 2, pp. 1–24, May 2020, doi:10.1145/3379542. DOI: 10.1145/3379542.

[111] Bertino, E. and Islam, N., "Botnets and Internet of Things Security," *Computer*, vol. 50, no. 2, pp. 76–79, Feb. 2017, doi:10.1109/mc.2017.62. DOI: 10.1109/mc.2017.62.

[112] Khan, M. A. and Salah, K., "IoT Security: Review, Blockchain Solutions, and Open Challenges," *Future Generation Computer Systems*, vol. 82, pp. 395–411, May 2018, doi:10.1016/j.future.2017.11.022. DOI: 10.1016/j.future.2017.11.022.

[113] Asharf, J. et al., "A Review of Intrusion Detection Systems Using Machine and Deep Learning in Internet of Things: Challenges, Solutions and Future Directions," *Electronics*, vol. 9, no. 7, p. 1177, Jul. 2020, doi:10.3390/electronics9071177. DOI: 10.3390/electronics9071177.

[114] Neto, E. C. P. et al., "CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment," *Sensors*, vol. 23, no. 13, p. 5941, Jun. 2023, doi:10.3390/s23135941. DOI: 10.3390/s23135941.

[115] STMicroelectronics, *Discovery kit with STM32H7S7L8 MCU*, https://www.st.com/en/evaluation-tools/stm32h7s78-dk.html. Accessed on May 29, 2025.

[116] Abbas, S. et al., "Evaluating Deep Learning Variants for Cyber-Attacks Detection and Multi-Class Classification in IoT Networks," *PeerJ Computer Science*, vol. 10, e1793, Jan. 2024, doi:10.7717/peerj-cs.1793. DOI: 10.7717/peerj-cs.1793.

[117] Abbas, S. et al., "A Novel Federated Edge Learning Approach for Detecting Cyberattacks in IoT Infrastructures," *IEEE Access*, vol. 11, pp. 112 189–112 198, 2023, doi:10.1109/ACCESS.2023.3318866. DOI: 10.1109/ACCESS.2023.3318866.

[118] Vajrobol, V. et al., "Adversarial Learning for Mirai Botnet Detection based on Long Short-Term Memory and XGBoost," *International Journal of Cognitive Computing in Engineering*, vol. 5, pp. 153–160, 2024, doi:10.1016/j.ijcce.2024.02.004. DOI: 10.1016/j.ijcce.2024.02.004.

[119] Gharaibeh, H. et al., "Deep Feature Extraction Framework Based on DNN for Enhancing Mirai Attachment Classification in Machine Learning," in *2023 2nd International Engineering Conference on Electrical, Energy, and Artificial Intelligence (EICEEAI)*, doi:10.1109/EICEEAI60672.2023.10590608, Zarqa, Jordan, 2023, pp. 1–5. DOI: 10.1109/EICEEAI60672.2023.10590608.

[120] Becerra-Suarez, F. L. et al., "Performance Evaluation of Deep Learning Models for Classifying Cybersecurity Attacks in IoT Networks," *Informatics*, vol. 11, no. 2, p. 32, May 2024, doi:10.3390/informatics11020032. DOI: 10.3390/informatics11020032.

[121] Aguru, A. D. and Erukala, S. B., "A Lightweight Multi-Vector DDoS Detection Framework for IoT-enabled Mobile Health Informatics Systems using Deep Learning," *Information Sciences*, vol. 662, p. 120 209, Mar. 2024, doi:10.1016/j.ins.2024.120209. DOI: 10.1016/j.ins.2024.120209.

[122] Kumar, A. G., Rastogi, A., and Ranga, V., "Evaluation of Different Machine Learning Classifiers on New IoT Dataset CICIoT2023," in *2024 International Conference on Intelligent Systems for Cybersecurity (ISCS)*, doi:10.1109/ISCS61804.2024.10581375, Gurugram, India, 2024, pp. 1–6. DOI: 10.1109/ISCS61804.2024.10581375.

[123] Saxe, J. and Berlin, K., "Deep Neural Network based Malware Detection using Two Dimensional Binary Program Features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, doi:10.1109/MALWARE.2015.7413680, Fajardo, PR, USA, 2015, pp. 11–20. DOI: 10.1109/MALWARE.2015.7413680.

[124] The imbalanced-learn developers, *RandomOverSampler*, https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html. Accessed on May 29, 2025.

[125] Thapa, S., Poudel, S., and Abouyoussef, M., "Tinyml-enabled intrusion detection for securing electric vehicle supply equipment (evse)," in *2025 1st International Conference on Secure IoT, Assured and Trusted Computing (SATC)*, IEEE, 2025, pp. 1–5.

[126] Buedi, E. D. et al., "Enhancing EV Charging Station Security Using a Multi-dimensional Dataset: CICEVSE2024," in *Data and Applications Security and Privacy XXXVIII*, doi:10.1007/978-3-031-65172-4_11, San Jose, CA, USA, 2024, pp. 171–190. DOI: 10.1007/978-3-031-65172-4_11.

[127] Arcot, S., Masum, M., Kader, M. S., Saha, A., and Chowdhury, M., "Tinyml for cybersecurity: Deploying optimized deep learning models for on-device threat detection on resource-constrained devices," in *2024 IEEE International Conference on Big Data (BigData)*, IEEE, 2024, pp. 5542–5550.

[128] Wang, W. et al., "Malware Traffic Classification using Convolutional Neural Network for Representation Learning," in *2017 International Conference on Information Networking (ICOIN)*, doi:10.1109/ICOIN.2017.7899588, Da Nang, Vietnam, 2017, pp. 712–717. DOI: 10.1109/ICOIN.2017.7899588.

[129] Sharma, A., Rani, S., and Shabaz, M., "An optimized stacking-based tinyml model for attack detection in iot networks," *Plos one*, vol. 20, no. 8, e0329227, 2025.

[130] Booij, T. M. et al., "ToN_IoT: The Role of Heterogeneity and the Need for Standardization of Features and Attack Types in IoT Network Intrusion Data Sets," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 485–496, 2022, doi:10.1109/JIOT.2021.3085194. DOI: 10.1109/JIOT.2021.3085194.

[131] Sengupta, J., Ruj, S., and Das Bit, S., "A Comprehensive Survey on Attacks, Security Issues and Blockchain Solutions for IoT and IIoT," *Journal of Network and Computer Applications*, vol. 149, p. 102 481, Jan. 2020, doi:10.1016/j.jnca.2019.102481. DOI: 10.1016/j.jnca.2019.102481.

[132] Syed, N. F. et al., "Denial of Service Attack Detection through Machine Learning for the IoT," *Journal of Information and Telecommunication*, vol. 4, no. 4, pp. 482–503, Jun. 2020, doi:10.1080/24751839.2020.1767484. DOI: 10.1080/24751839.2020.1767484.

[133] Pirayesh, H., Kheirkhah Sangdeh, P., and Zeng, H., "Securing ZigBee Communications Against Constant Jamming Attack Using Neural Network," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4957–4968, 2021, doi:10.1109/JIOT.2020.3034128. DOI: 10.1109/JIOT.2020.3034128.

[134] Balueva, A., Desnitsky, V., and Ushakov, I., "Approach to Detection of Denial-of-Sleep Attacks in Wireless Sensor Networks on the Base of Machine Learning," in *Studies in Computational Intelligence*. Springer International Publishing, Oct. 2019, pp. 350–355, doi:10.1007/978-3-030-32258-8_41. DOI: 10.1007/978-3-030-32258-8_41.

[135] Yan, J. et al., "A Novel Sybil Attack Detection Scheme in Mobile IoT based on Collaborate Edge Computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2023, no. 1, Mar. 2023, doi:10.1186/s13638-023-02233-8. DOI: 10.1186/s13638-023-02233-8.

[136] The imbalanced-learn developers, *RandomUnderSampler*, https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html. Accessed on May 29, 2025.

[137] scikit-learn, *PowerTransformer*, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html. Accessed on May 29, 2025.

[138] SciPy Core Developer, *One-way ANOVA tests*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html. Accessed on May 29, 2025.

[139] Keras Google Group, *KerasTuner*, https://keras.io/keras_tuner/. Accessed on May 29, 2025.

[140] McHugh, M. L., "Interrater Reliability: The Kappa Statistic," *Biochemia Medica*, pp. 276–282, 2012, doi:10.11613/bm.2012.031. DOI: 10.11613/bm.2012.031.

[141] Abadi, M. et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv preprint arXiv:1603.04467*, vol. 1, no. 1, pp. 1–19, 2016, doi:10.48550/arXiv.1603.04467. DOI: 10.48550/arXiv.1603.04467.

[142] Wu, J. et al., "Quantized Convolutional Neural Networks for Mobile Devices," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, doi:10.1109/CVPR.2016.521, Los Alamitos, CA, USA, Jun. 2016, pp. 4820–4828. DOI: 10.1109/CVPR.2016.521.

[143] Mazinani, A., Davoli, L., Pau, D. P., and Ferrari, G., "Air Quality Estimation with Embedded AI-Based Prediction Algorithms," in *2023 International Conference on Information Technology Research and Innovation (ICITRI)*, doi:10.1109/ICITRI59340.2023.10249864, Jakarta, Indonesia, Aug. 2023, pp. 87–92. DOI: 10.1109/ICITRI59340.2023.10249864.

[144] STMicroelectronics, *Cube-ai*, https://stm32ai.st.com/stm32-cube-ai, Accessed: 2025-05-29.

[145] Burrello, A. et al., "Tcn mapping optimization for ultra-low power time-series edge inference," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2021, pp. 1–6. DOI: 10.1109/ISLPED52811.2021.9502494.

[146] Duguma, A. L. and Bai, X., "How the internet of things technology improves agricultural efficiency," *Artificial Intelligence Review*, vol. 58, no. 2, p. 63, 2024.

[147] Chen, Z., Sivaparthipan, C., and Muthu, B., "Iot based smart and intelligent smart city energy optimization," *Sustainable Energy Technologies and Assessments*, vol. 49, p. 101 724, 2022.

[148] Humayun, M., Alsaqer, M. S., and Jhanjhi, N., "Energy optimization for smart cities using iot," *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2 037 255, 2022.

[149] García, L., Parra, L., Jimenez, J. M., Lloret, J., and Lorenz, P., "Iot-based smart irrigation systems: An overview on the recent trends on sensors and iot systems for irrigation in precision agriculture," *Sensors*, vol. 20, no. 4, p. 1042, 2020.

[150] Dhanaraju, M., Chenniappan, P., Ramalingam, K., Pazhanivelan, S., and Kaliaperumal, R., "Smart farming: Internet of things (iot)-based sustainable agriculture," *Agriculture*, vol. 12, no. 10, p. 1745, 2022.

[151] Bhardwaj, V., Joshi, R., and Gaur, A. M., "Iot-based smart health monitoring system for covid-19," *SN Computer Science*, vol. 3, no. 2, p. 137, 2022.

[152] Kishor, A. and Chakraborty, C., "Artificial intelligence and internet of things based healthcare 4.0 monitoring system," *Wireless personal communications*, vol. 127, no. 2, pp. 1615–1631, 2022.

[153] Sokullu, R., Akkaş, M. A., and Demir, E., "Iot supported smart home for the elderly," *Internet of Things*, vol. 11, p. 100 239, 2020.

[154] Stolojescu-Crisan, C., Crisan, C., and Butunoi, B.-P., "An iot-based smart home automation system," *Sensors*, vol. 21, no. 11, p. 3784, 2021.

[155] Mohapatra, H., Rath, A. K., and Panda, N., "Iot infrastructure for the accident avoidance: An approach of smart transportation," *International Journal of Information Technology*, vol. 14, no. 2, pp. 761–768, 2022.

[156] Karim, A., "Development of secure internet of vehicle things (iovt) for smart transportation system," *Computers and Electrical Engineering*, vol. 102, p. 108 101, 2022.

[157] Davoli, L., Veltri, L., Ventre, P. L., Siracusano, G., and Salsano, S., "Traffic Engineering with Segment Routing: SDN-based Architectural Design and Open Source Implementation," in *2015 Fourth European Workshop on Software Defined Networks (EWSDN)*, IEEE, Sep. 2015, pp. 111–112. DOI: 10.1109/EWSDN.2015.73.

[158] Salsano, S., Veltri, L., Davoli, L., Ventre, P. L., and Siracusano, G., "PMSR–Poor Man's Segment Routing, a Minimalistic Approach to Segment Routing and a Traffic Engineering Use Case," in *2016 IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Apr. 2016, pp. 598–604. DOI: 10.1109/NOMS.2016.7502864.

[159] Ja'afreh, M. A., Adhami, H., Alchalabi, A. E., Hoda, M., and El Saddik, A., "Toward integrating software defined networks with the internet of things: A review," *Cluster Computing*, pp. 1–18, 2022.

[160] Ezechi, C., Akinsolu, M. O., Sangodoyin, A. O., Akinsolu, F. T., and Sakpere, W., "Software-defined networking in cyber-physical systems: Benefits, challenges, and opportunities," *Cyber Physical System 2.0*, pp. 44–69, 2025.

[161] Braun, W. and Menth, M., "Software-defined networking using openflow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[162] Polónio, J., Moura, J., and Marinheiro, R. N., "On the road to proactive vulnerability analysis and mitigation leveraged by software defined networks: A systematic review," *IEEE Access*, 2024.

[163] Elsayed, M. S., Le-Khac, N.-A., and Jurcut, A. D., "Insdn: A novel sdn intrusion dataset," *IEEE access*, vol. 8, pp. 165 263–165 284, 2020.

[164] Mashal, I., Alsaryrah, O., Chung, T.-Y., Yang, C.-Z., Kuo, W.-H., and Agrawal, D. P., "Choices for interaction with things on internet and underlying issues," *Ad Hoc Networks*, vol. 28, pp. 68–90, 2015.

[165] Burhan, M., Rehman, R. A., Khan, B., and Kim, B.-S., "Iot elements, layered architectures and security issues: A comprehensive survey," *sensors*, vol. 18, no. 9, p. 2796, 2018.

[166] Miloslavskaya, N. and Tolstoy, A., "IoTBlockSIEM for Information Security Incident Management in the Internet of Things Ecosystem," *Cluster Computing*, vol. 23, no. 3, pp. 1911–1925, Jun. 2020, doi:10.1007/s10586-020-03110-5. DOI: 10.1007/s10586-020-03110-5.

[167] Open Networking Foundation, *OpenFlow Switch Specification*.

[168] Amin, R., Reisslein, M., and Shah, N., "Hybrid sdn networks: A survey of existing approaches," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3259–3306, 2018.

[169] Elsayed, R. A., Hamada, R. A., Abdalla, M. I., and Elsaid, S. A., "Securing iot and sdn systems using deep-learning based automatic intrusion detection," *Ain Shams Engineering Journal*, vol. 14, no. 10, p. 102 211, 2023.

[170] Kingma, D. P. and Ba, J., "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, vol. 1, no. 9, pp. 1–15, 2014. DOI: 10.48550/arXiv.1412.6980.

[171] Hnamte, V., Najar, A. A., Nhung-Nguyen, H., Hussain, J., and Sugali, M. N., "Ddos attack detection and mitigation using deep neural network in sdn environment," *Computers & Security*, vol. 138, p. 103 661, 2024.

[172] Ibrahim, S., Youssef, A. M., Shoman, M., and Taha, S., "Intelligent sdn to enhance security in iot networks," *Egyptian Informatics Journal*, vol. 28, p. 100 564, 2024.

[173] Khanal, B., Kumar, C., and Ansari, M. S. A., "Real-time anomaly detection framework to mitigate emerging threats in software defined networks," *Journal of Network and Systems Management*, vol. 33, no. 2, p. 26, 2025.

[174] El Sayed, M. S., Le-Khac, N.-A., Azer, M. A., and Jurcut, A. D., "A flow-based anomaly detection approach with feature selection method against ddos attacks in sdns," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 4, pp. 1862–1880, 2022.

[175] Said, R. B., Sabir, Z., and Askerzade, I., "Cnn-bilstm: A hybrid deep learning approach for network intrusion detection system in software-defined networking with hybrid feature selection," *IEEE Access*, vol. 11, pp. 138 732–138 747, 2023.

[176] Cui, M., Chen, J., Qiu, X., Lv, W., Qin, H., and Zhang, X., "Multi-class intrusion detection system in sdn based on hybrid bilstm model," *Cluster Computing*, vol. 27, no. 7, pp. 9937–9956, 2024.

[177] Metasploit, *Metasploitable2*, https://www.vulnhub.com/entry/metasploitable-2,29/. Accessed on March 10, 2025.

[178] Linux Foundation Collaborative Project, *Open vSwitch*.

[179] Open Networking Foundation, *Open Network Operating System (ONOS)*.

[180] Lantz, B., Heller, B., and McKeown, N., "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX, Monterey, California, 2010. DOI: 10.1145/1868447.1868466.

[181] scikit-learn community, *LabelEncoder*, [Accessed 31-March-2025]. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[182] scikit-learn community, *Tree-based_feature_selection*, [Accessed 31-March-2025]. [Online]. Available: https://scikit-learn.org/stable/modules/feature_selection.html.

[183] scikit-learn developers, *StandardScaler*.

[184] Ahsan, M. M., Mahmud, M. P., Saha, P. K., Gupta, K. D., and Siddique, Z., "Effect of data scaling methods on machine learning algorithms and model performance," *Technologies*, vol. 9, no. 3, p. 52, Jul. 2021. [Online]. Available: https://doi.org/10.3390/technologies9030052.

[185] Wikipedia contributors, *Hyperparameter optimization — Wikipedia, The Free Encyclopedia*, [Accessed 31-March-2025], 2025. [Online]. Available: https://en.wikipedia.org/wiki/Hyperparameter_optimization.

[186] TensorFlow, *Post-training quantization*, [Accessed 31-March-2025]. [Online]. Available: https://www.tensorflow.org/model_optimization/guide/quantization/post_training.

[187] Han, S., Pool, J., Tran, J., and Dally, W., "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[188] McHugh, M. L., "Interrater Reliability: The Kappa Statistic," *Biochemia Medica*, pp. 276–282, 2012. DOI: 10.11613/bm.2012.031.

[189] STMicroelectronics, *NUCLEO-G474RE – STM32 Nucleo-64 development board with STM32G474RE MCU, supports Arduino and ST morpho connectivity –*.