CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	A scalable security approach for IoT networks: Blockchain and anomaly-based IDS on hardware accelerators
Authors	Ngo, Duc-Minh
Publication date	2024
Original Citation	Ngo, D-M. 2024. A scalable security approach for IoT networks: Blockchain and anomaly-based IDS on hardware accelerators. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2024, Duc-Minh Ngo. - https://creativecommons.org/licenses/by/4.0/
Download date	2026-11-17 04:48:28
Item downloaded from	https://hdl.handle.net/10468/17907

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Ollscoil na hÉireann, Corcaigh

National University of Ireland, Cork



A Scalable Security Approach for IoT Networks: Blockchain and Anomaly-based IDS on Hardware Accelerators

Thesis presented by

DUC-MINH NGO

[0000-0002-9253-8777]

for the degree of

Doctor of Philosophy

University College Cork

School of Engineering

Department of Electrical and Electronic Engineering

Head of School: Prof. Jorge Oliveira

Supervisors: Dr. Emanuel Popovici
Dr. Colin C. Murphy
Prof. Dr. Andriy Temko

2024

Contents

List of Figures . . . iv
List of Tables . . . vii
List of Acronyms . . . ix
List of Publications . . . xii
Competitions . . . xiii
Acknowledgements . . . xiv
ABSTRACT . . . xv

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Literature Review . . . 3
1.2.1 IoT Security: An Architecture View . . . 3
1.2.2 IoT Security on The Cloud: Smart Contract Technology for IoT Access Control . . . 5
1.2.3 IoT Security at The Edge: Anomaly-based IDSs Using Hardware Accelerator . . . 9
1.3 Aims of Thesis . . . 12
1.4 Thesis Outline . . . 12
References . . . 15

2 Intrusion Detection Systems **20**
2.1 Network Architecture . . . 20
2.1.1 TCP/IP and OSI Models . . . 20
2.1.2 Packet Structure . . . 22
2.1.3 Network Attacks . . . 22
2.2 Intrusion Detection System . . . 24
2.2.1 Signature-based IDS . . . 24
2.2.2 Anomaly-based IDS . . . 25
2.2.2.1 Supervised Learning Approach . . . 26
2.2.2.2 Unsupervised Learning Approach . . . 27
2.3 Datasets . . . 27
2.3.1 IoT-23 . . . 29
2.3.2 UNSW-NB15 . . . 30
2.4 Artificial Neural Network for Anomaly-based IDSs . . . 31
2.4.1 Methodology . . . 31
2.4.1.1 Artificial Neural Networks . . . 31
2.4.1.2 Dataset . . . 33
2.4.1.3 Related Work . . . 34
2.4.2 Proposed Architecture . . . 35
2.4.2.1 Overview . . . 35
2.4.2.2 Data preparation . . . 36
2.4.2.3 Training . . . 38
2.4.2.4 Offline Inference . . . 39
2.5 Summary . . . 40
References . . . 42

3 Leveraging Hardware Accelerators for Enhancing Security in IoT Networks **45**
3.1 Background Knowledge . . . 46
3.1.1 IoT Architecture . . . 47
3.1.2 IoT Protocols . . . 49
3.1.3 Attacks in IoT . . . 50
3.2 Hardware Accelerators . . . 51
3.2.1 Platform Comparison . . . 52
3.2.2 Max78000 AI-assisted Microcontroller . . . 54
3.2.3 PYNQ-Z2 FPGA SoC . . . 54
3.2.4 Alveo U280 FPGA Card . . . 55
3.3 An Anomaly-based IDS Framework using Hardware Accelerators 55
3.3.1 Related Works . . . 56
3.3.2 Evaluation Metrics . . . 58
3.3.3 HH-NIDS Framework . . . 59
3.3.4 Implementation . . . 61
3.3.5 Results . . . 67
3.3.5.1 Implemented Result . . . 67
3.3.5.2 Accuracy . . . 68
3.3.5.3 Performance . . . 73
3.3.6 Discussion . . . 76
3.4 Summary . . . 77
References . . . 79

4 Deep Packet Inspection: A Convolutional Neural Networks Approach **84**
4.1 Background and Related Work . . . 85
4.1.1 Deep Packet Inspection . . . 85
4.1.2 The IoT-23 Dataset . . . 90
4.1.3 Related Work . . . 91
4.2 Proposed Approach . . . 94
4.2.1 Overview . . . 94
4.2.2 Implementation . . . 94
4.3 Experiment . . . 97
4.3.1 Accuracy . . . 97
4.3.2 Performance . . . 99
4.4 Summary . . . 100
4.4.1 Optimise CNN models . . . 101
References . . . 104

5 Distributed-ledger Technology: Blockchain and Smart Contracts for IoT Security **107**
5.1 Distributed Ledger Technology: Platforms and Applications . . . 108
5.1.1 Blockchain and Consensus Mechanisms . . . 108
5.1.2 Bitcoin Blockchain: Structure and Operations . . . 110
5.1.3 Ethereum Blockchain: Smart Contracts . . . 112
5.1.4 Hyperledger Blockchain . . . 114

5.1.5 Decentralised Applications: State-of-the-art Platforms . . 115
5.2 Blockchain and IoT Security 117
5.2.1 Blockchain and Anomaly Detection 117
5.2.2 Smart Contract in IoT Access Control 119
5.3 Smart Contracts in IoT Security: Design, Implementation, and Evaluation 120
5.3.1 Methodology 121
5.3.2 Implementation 122
5.3.3 Evaluation 126
5.3.4 Discussion 128
5.4 Summary 129
References 131

6 Framework Design and Implementation **134**
6.1 Anomaly-based IDS on Hardware Platforms 135
6.1.1 Data Pre-processing 135
6.1.2 Training and Validation 136
6.1.3 Inference Engines 136
6.2 Proposed Architecture 136
6.2.1 Overview 136
6.2.2 Implementation 138
6.2.2.1 Web3 Applications 138
6.2.2.2 Gateways 140
6.2.2.3 Packet Processor 141
6.2.3 Design Verification 142
6.2.3.1 ANN model 143
6.2.3.2 CNN model 145
6.3 Experiments 146
6.3.1 Setup 146
6.3.2 FPGA-based Gateway 147
6.3.2.1 Implementation Results 147
6.3.2.2 Performance 148
6.3.3 Smart Contract 149
6.3.3.1 Transaction Cost 149
6.3.3.2 Device Verification 150
6.3.4 Platforms Comparison 152
6.3.5 FPGA Reliability 155
6.4 Discussion 156
6.4.1 The Cloud Layer 156
6.4.2 The Control Layer 156
6.4.3 The IoT Devices Layer 157
6.5 Summary 157
References 159

7 Conclusion and Future Work **160**
7.1 Scalable Blockchain Solutions 162
7.2 Anomaly-based IDSs on Hardware Accelerators 163

List of Figures

1.1 A high-level view of an IoT architecture with four layers: application, processing, network, and perception. This architecture is an abstraction of the practical IoT topology and application architecture on the left-hand side. 4

2.1 The TCP/IP model compared with the OSI model (Fiberbit 2013). 21
2.2 The packet header structure in the Ethernet II frame standard (Huawei 2024). 22
2.3 A three layers ANN model. 32
2.4 System overview in training, hierarchical decision-making, and inference phases using ANN on the IoT-23 dataset. 33
2.5 The process flow of the proposed anomaly-based IDS framework in four phases: Data preparation, Training (offline) and Inferences (offline and online). 36
2.6 The data flow modelling of the Data Preparation phase. 37
2.7 GPU-based acceleration for training NN models. 38
2.8 Hierarchical decision making flow-chart 39

3.1 The evolution of IoT architecture in four phases. 46
3.2 The Three-layer model. 48
3.3 The Five-layer and Seven-layer models from Cisco Systems. . . 49
3.4 The relation between flexibility and power efficiency of CPUs, GPUs, FPGAs and ASICs when dealing with different tasks. . . . 53
3.5 The proposed HH-NIDS framework architecture. 59
3.6 The HH-NIDS framework's processing flow from training to hardware implementations. 61
3.7 Neural network inference acceleration on Zynq-7000 SoC architecture. 63
3.8 The generated FPGA block architecture using the HLS implementation approach. 64
3.9 Pipeline implementation of the *neural Network Architecture* block on an FPGA-based architecture includes four phases: Input buffering (P1), Layer_0 calculation (P2), Layer_1 calculation (P3), and Output buffering (P4). 65
3.10 The *MULx11* block calculation architecture. 66
3.11 The trained results in fifty epochs with three different learning rates: 0.0005, 0.001, and 0.01 for the MAX78000 microcontroller. 70
3.12 The trained results in fifty epochs with three different learning rates: 0.0005, 0.001, and 0.01 for the PYNQ-Z2 SoC FPGA. . . 71
3.13 The pre-process, inference, and post-process time from different implementations (except the MAX78000EVKIT, see text). . 74
3.14 The CPU, GPU, and FPGA inference times for different input buffer sizes. 74
3.15 The wareform simulation results from the NN on FPGA using a Verilog approach. 75

4.1 The proposed framework in (Lotfollahi et al. 2020) combining SAE and CNN models for traffic classification. 87
4.2 The Packet2Vec approach's process flow. 88
4.3 The featureless modelling approaches using 1D-CNN. 90
4.4 The IoT-23 dataset's labelling process flow 91
4.5 The proposed approach's process flow using the HH-NIDS framework (Ngo, Lightbody, Temko, Pham-Quoc, et al. 2023a) on the IoT-23 dataset. 94
4.6 HH-NIDS framework architecture 95
4.7 The four stages in the process flow, namely Training (ANN), Labelling, Training (CNN), and Inference 96
4.8 The trained results on the IoT-23 dataset with different learning rates 98
4.9 Feature extraction and inference time on CPU and GPU for different input buffer sizes. 100

5.1 Bitcoin Blockchain using double SHA-256 hashing technique to generate the block hash. 110
5.2 Bitcoin Blockchain operation from the user point of view. 112
5.3 The smart contract development and interaction process flow. . 113
5.4 Hyperledger Blockchain frameworks' evolution. 114
5.5 Architecture of the proposed HybridChain framework (Sharadqh et al. 2023). 117
5.6 The proposed Blockchain-based IoT network architecture with three layers: *IoT Devices*, *Control*, and *Cloud*. 121
5.7 The first prototype architecture block diagram. 122
5.8 The smart contract UML class diagram. 123
5.9 Adding a new admin sequence diagram. 124
5.11 The sequence diagram of device verification process. 125
5.10 Adding a new device sequence diagram. 125
5.12 The simulation environment with ganache Blockchain running on localhost. 126
5.13 The generated topology in the first scenario with the data from the Blockchain. 127

6.1 The platform-dependent anomaly-based IDS framework on the IoT-23 dataset. 135
6.2 The proposed architecture with three layers: *IoT Devices*, *Control* and *Cloud*. 137
6.3 The sequence diagram for adding a new administrator (denoted as (A)) process. 138
6.4 The sequence diagram for adding a new device (denoted as D) to the network. 139
6.5 The block diagram of the NetFPGA-PLUS Gateway with security integration. 140
6.6 The *P2I processor* module state machines and data flow. 141

6.7 The *Packet Selector* module's flow chart expressing packet priority order. 142
6.8 Two stages of the design verification process on an ANN model. 144
6.9 Three stages of the design verification process on a CNN model. 145
6.10 The setup with the full system in the physical network. 146
6.11 The performance test results for different packet sizes (64B-1500B). 149
6.13 The connection topology generated from Blockchain data in the first scenario. 151
6.12 The device verification time on the smart contract in different scenarios. 151
6.14 The inference results between engines in the first scenario. . . . 153
6.16 The inference results between engines in the third scenario. . . 154
6.15 The inference results between engines in the second scenario. . 154

7.1 Future work on enhancing scalable security in the proposed system. 162

List of Tables

1.1 Summary of existing literature on IoT security and access control using Blockchain and smart contracts. 6
1.2 Summary of anomaly-based IDS for IoT networks 10

2.1 The common datasets in the field. 27
2.2 The IoT-23 dataset record distribution 30
2.3 USNW-NB15 dataset record distribution 31
2.4 Sixteen labelled flows in the IoT-23 dataset, divided into five groups 34
2.5 Inference results (in %) of the selected plans on the IoT-23 dataset 37
2.6 Result comparison to related works in the IoT-23 dataset 40

3.1 Summary of anomaly-based IDS on the IoT-23 dataset 57
3.2 Flow Features from the IoT-23 and UNSW-NB15 datasets 62
3.3 The IoT-23 dataset implemented results on PYNQ-Z2 SoC FPGA 68
3.4 The UNSW-NB15 dataset implemented results on PYNQ-Z2 SoC FPGA 69
3.5 The inference results (in %) for the two trained models on the MAX78000EVKIT. 72
3.6 The inference results (in %) for the two trained models on the PYNQ-Z2 board. 72
3.7 Result comparison to related works in the IoT-23 dataset 72
3.8 Comparison in inference time on different platforms and between related works 76

4.1 Summary of state-of-the-art DPI systems. 92
4.2 Summary of anomaly-based IDS on the IoT-23 dataset 93
4.3 The implemented 2D-CNN model structure 97
4.4 Inference accuracy and F1-score comparison to related work . . 99
4.6 The CNN model structure in the fifth trial. 102
4.5 The training and validation results of different CNN models (C: Convolution layer, P: Pooling layer, D: Dense layer). 102

5.1 Summary of popular consensus mechanisms in the Blockchain platforms. 111
5.2 Smart contract technologies and platforms. 116
5.3 The time for querying block history and verifying devices on the Blockchain. 127
5.4 The total transaction time and gas cost when adding more devices. 128

6.1 The implemented results of FPGA hardware on Vivado 2018.2. 148
6.2 Transaction time and gas used of functions in the smart contract. 150
6.3 The total transaction time and gas cost when adding more devices. 152
6.4 The pcap file's information. 152
6.5 The routing information programmed on the FPGA Gateway. . . 153
6.6 FPGA Reliability results of ANN and CNN models. 155

I, DUC-MINH NGO, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere. All the assistance received in preparing this thesis and sources have been acknowledged. I have read and understood the regulations of University College Cork concerning plagiarism and intellectual property.

DUC-MINH NGO

List of Acronyms

1D-CNN One-dimensional Convolutional Neural Network.

2D-CNN Two-dimensional Convolutional Neural Network.

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks.

AI Artificial Intelligence.

ANN Artificial Neural Network.

ASIC Application-Specific Integrated Circuit.

CNN Convolutional Neural Network.

CoAP Constrained Application Protocol.

Dapp Decentralised Applications.

DDOS Distributed Denial of Service.

DDS Data Distribution Service.

DL Deep Learning.

DLT Distributed-ledger Technology.

DNN Deep Neural Network.

DOS Denial of Service.

DPI Deep Packet Inspection.

DPoS Delegated Proof of Stake.

DT Decision Tree.

DV Design Verification.

FPGA Field-Programmable Gate Array.

GNB Gaussian Naive-Bayes.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

HDL Hardware Description Language.

HLS High-Level Synthesis.

HPC High-Performance Computing.

HTTP Hypertext Transfer Protocol.

IDS Intrusion Detection System.

IoT Internet of Thing.

IP Internet Protocol.

KNN K-Nearest Neighbor.

LR Linear Regression.

LSTM Long Short-Term Memory.

LTE Long-Term Evolution.

MitM Man-in-the-Middle.

ML Machine Learning.

MQTT Message Queuing Telemetry Transport.

MTU Maximum Transmission Unit.

NB Naive Bayes.

NB-IoT Narrowband Internet of Things.

NN Neural Network.

NPoS Nominated Proof of Stake.

OSI Open Systems Interconnection.

PBFT Practical Byzantine Fault Tolerance.

PCA Principal Component Analysis.

PoA Proof of Authority.

PoB Proof of Burn.

PoET Proof of Elapsed Time.

PoH Proof of History.

ReLU Rectified Linear Unit.

RF Random Forest.

RTL Register Transfer Level.

SAE Stacked Autoencoder.

SVM Support Vector Machine.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UTXO Unspent Transaction Output.

UVM Universal Verification Methodology.

XML eXtensible Markup Language.

List of Publications

Parts of this work have previously been published in scientific papers below:

- **Duc-Minh Ngo**, Dominic Lightbody, Andriy Temko, Colin C Murphy, Emanuel Popovici. **A Scalable Security Approach in IoT Networks: Smart Contracts and Anomaly-based IDS for Gateways using Hardware Accelerators**. In: IEEE Access 2024.
- **Duc-Minh Ngo**, Dominic Lightbody, Andriy Temko, Colin C Murphy, Emanuel Popovici. **Enhancing Security and Scalability in IoT Networks Applying Blockchain Technology**. In: Cooperative Design, Visualization, and Engineering: 21st International Conference, CDVE 2024, Valencia, Spain. Proceedings (p. 265). Springer Nature.
- **Duc-Minh Ngo**, Dominic Lightbody, Andriy Temko, Cuong Pham-Quoc, Ngoc-Thinh Tran, Colin C Murphy, Emanuel Popovici. **Network Attack Detection on IoT Devices Using 2D-CNN Models**. In: International Conference on Intelligence of Things (pp. 237-247). Cham: Springer Nature Switzerland.
- **Duc-Minh Ngo**, Dominic Lightbody, Andriy Temko, Cuong Pham-Quoc, Ngoc-Thinh Tran, Colin C Murphy, Emanuel Popovici. **HH-NIDS: heterogeneous hardware-based network intrusion detection framework for IoT security**. In: Future Internet, 15(1), 9.
- **Duc-Minh Ngo**, Andriy Temko, Colin C Murphy, Emanuel Popovici. **FPGA Hardware Acceleration Framework for Anomaly-based Intrusion Detection System in IoT**. 2021 31st International Conference on Field-Programmable Logic and Applications (FPL) (pp. 69-75). IEEE.

Co-author:

- Dominic Lightbody, **Duc-Minh Ngo**, Andriy Temko, Colin C Murphy, Emanuel Popovici. **Dragon_Pi: IoT Side-Channel Power Data Intrusion Detection Dataset and Unsupervised Convolutional Autoencoder for Intrusion Detection**. In: Future Internet 16 (3), 88.
- Dominic Lightbody, **Duc-Minh Ngo**, Andriy Temko, Colin C Murphy, Emanuel Popovici. **Attacks on IoT: side-channel power acquisition framework for intrusion detection**. In: Future Internet 15 (5), 187.
- Dominic Lightbody, **Duc-Minh Ngo**, Andriy Temko, Colin C Murphy, Emanuel Popovici. **Host-based intrusion detection system for iot using convolutional neural networks**. In: 2022 33rd Irish Signals and Systems Conference (ISSC), 1-7.

Duc-Minh Ngo

Competitions

- **MAX78000 AI Design Contest**[1]: In 2021, we were honoured to win the first prize in the prestigious MAX78000 AI Design Contest, a global competition highlighting innovative applications of AI in embedded systems.

  My contribution focused on implementing and accelerating artificial neural network (ANN) models on the MAX78000 platform, a state-of-the-art AI-assisted microcontroller known for its ultra-low power consumption. By leveraging the MAX78000's hardware-accelerated capabilities, I successfully implemented and evaluated ANN models on the platform, showcasing its potential in low-power, high-performance anomaly detection applications. The team members: Duc-Minh Ngo, Dominic Lightbody, and Cathal Kelly.

- **Kaggle AI competition**[2]: In 2022, we won the bronze medal in the Sartorius - Cell Instance Segmentation competition hosted on Kaggle, a high-level AI and data science challenge platform.

  This competition focused on image processing using deep learning models for accurately segmenting cells in microscopic images. This achievement demonstrated our ability to tackle complex real-world problems in AI and image processing. The team members: Duc-Minh Ngo, Dominic Lightbody, Emanuel Popovici, Andriy Temko.

[1] https://www.elektormagazine.com/ai-contest-max78000
[2] https://www.kaggle.com/competitions/sartorius-cell-instance-segmentation

Acknowledgements

Firstly, I would like to express my sincere gratitude to my lead supervisor, *Dr. Emanuel Popovici* for his warm support, taking the time from the beginning of my work to orient the research, and also during the step-by-step of the thesis process. It reminds me of the special period back in September 2020 when I first arrived in Ireland; the country was in a lockdown state due to Covid-19. We have gone through a tough time together that has never happened before. Besides, we have learned and achieved remarkable milestones in both life and work together. He teaches me life-changing lessons and gives me opportunities to approach jobs in semiconductor leading companies. More importantly, I have learned from him to have a positive attitude in any circumstances.

Secondly, I would also like to thank my two other supervisors, *Dr. Colin Murphy* and *Prof. Dr. Andriy Temko* in different parts of my PhD journey. I have learned machine learning from *Prof. Dr. Andriy Temko*, his recommendations are useful in improving the thesis and publications. Especially my sincere thanks go to *Dr. Colin Murphy*, who has guided me and improved my skills a lot while writing research papers and reports. Besides, he always gives useful advice not only in work but also for life balancing.

To the Embedded System labmates, I would like to thank *Dominic Lightbody, Feargal Conall O'Sullivan, Sergi Gómez Quintana* and the many other individuals for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last four years.

Last but not least, I am grateful to God for the good health and well-being that were necessary to complete this degree. I would like to say thank you so much to my family. I could not go far on my way without their support and encouragement.

The degree is supported in part by a grant from Science Foundation Ireland INSIGHT Centre for Data Analytics (Grant number 12/RC/2289-P2), which is co-funded under the European Regional Development Fund. Support from Qualcomm through a philanthropic gift is also acknowledged. We would also like to acknowledge the support from Analog Devices, AMD/Xilinx, and Dell for various parts of this work.

This thesis acknowledges the indispensable role that "ChatGPT" and "Grammarly" played in the development of this study. Their assistance has proven extremely beneficial, enhancing the overall quality and depth of the research. We used these tools for language refinement, grammar checking, and enhancing the clarity of writing.

ABSTRACT

Cybersecurity plays a crucial role in today's interconnected world as cyber threats continue to rise. Moreover, with the ever-increasing dependence on digital infrastructure across industries, organizations must develop robust cybersecurity measures to detect and mitigate network attacks. This thesis studies Machine Learning and Distributed-ledger Technology to detect and mitigate network attacks in IoT networks using hardware accelerator platforms.

This study researches a scalable security approach for IoT networks from the edge to the cloud, combining Blockchain technology and anomaly-based Intrusion Detection Systems. From the top-down view, the cloud layer operates a Blockchain network containing smart contracts for securely managing IoT devices. At the device layer, machine learning models are trained to detect network attacks using hardware accelerators, including GPUs (Graphics Processing Units) and FPGAs (Field-Programmable Gate Arrays). The research highlights three main key findings. Firstly, smart contracts can manage network states efficiently and consistently by recording events on a distributed ledger. The data on a Blockchain are immutable and accessible from the Internet. The Blockchain's key contribution is to ensure data integrity, providing scalability and flexibility to the network at an abstract level. Secondly, machine learning models can detect unseen attacks and achieve high detection rates. Thirdly, lightweight machine learning models can be accelerated on FPGAs, significantly reducing overhead and adapting in high-speed, real-time network environments with minimum latency.

The first implemented prototype has achieved accuracies of 98.57% and 99.66% on the UNSW-NB15 and IoT-23 datasets, respectively. The inference engine was tested on different platforms: CPU, GPU, AI-microcontroller, and FPGA. In particular, the machine learning models are optimized on FPGA, resulting in a speedup compared to other platforms. The prototype FPGA-based Gateway was deployed on the Alveo U280 card (NetFPGA-PLUS) to evaluate the proposed architecture. The artificial neural network models have achieved the highest processing speed at 20Gbps with a packet loss rate of approximately 0.13%. The convolutional neural network models have produced a maximum processing speed of 8.38 Mbps, corresponding to 16,895 packets per second. The proposed approach offers innovative, scalable security solutions for IoT networks that can be mapped to the latest computing architectures.

Chapter 1

Introduction

1.1 Motivation

Internet of Things (IoT) security has become increasingly important in the 2020s due to its remarkable technological evolution. Firstly, there are ever-increasing demands for automation and smart devices in essential sectors, including healthcare, military, manufacturing and smart homes/cities, to name a few. This leads to an estimation of billions of new IoT devices joining each year (Lionel Sujay Vailshery 2024). Secondly, Artificial Intelligence (AI) technology is having a significant impact on human life. For example, a machine can imitate human behaviours by learning from experiences and then being able to evolve to deal with complicated tasks very rapidly; thus, hackers can use AI to explore a system's vulnerabilities or attack the system to disrupt the system's operations. Additionally, while the overall IoT architecture generally follows network infrastructure standards, the number of layers, technologies, and protocols within each layer are diverse. Building a secure and scalable IoT network requires attention to every aspect.

In 2024, the financial market witnessed a new wave with the spot Exchange-Traded Fund (ETF) approvals (Gary Gensler 2024) of the two most popular cryptocurrencies, Bitcoin and Ethereum. These approvals have made cryptocurrencies becoming more valuable and accessible than before[1]. Blockchain, the underlying technology behind cryptocurrencies, has gained significant adoption

[1]Disclaimer: Parts of this thesis study Distributed-ledger Technology, Blockchain and smart contract platforms associated with digital assets (cryptocurrencies). The content presented in this thesis is for educational purposes only. It is not intended to serve as financial advice, investment recommendations, or endorse any specific technologies, platforms, or digital assets.

with numerous emerging applications. Specifically, Blockchain is a form of DLT that operates a decentralised, peer-to-peer network to maintain synchronised copies of the ledger. This distributed data storage ensures data integrity, redundancy and security by eliminating the single point of failure in the centralised storage systems. Smart contracts were introduced in modern Blockchain networks, which have the ability to integrate business logic into the Blockchain. Smart contracts are triggered by transactions and execute automatically without a central entity. This work utilises smart contracts to manage access control in IoT networks.

In the meantime, the development of AI has dramatically impacted many areas, including the shift in implementing Intrusion Detection Systems (IDSs). AI applications have become a prominent topic because this technology can train machines to mimic human cognitive functions. It has achieved success in solving many practical problems, such as speech-recognition (Gao et al. 2019), encryption (Maitra et al. 2019), biosignal monitoring (Antonopoulos and Voros 2017), etc. Besides, Machine Learning (ML) (Expertsystem 2022) is renowned as an application of AI that can improve itself based on experience. ML has shown high efficiency and applicable results in predicting attacks in anomaly-based IDS, which is one of the classification problems. Many ML classification techniques (Mandy Sidana 2017) such as linear classifiers, logistic regression, support vector machines (SVM), decision trees (DT) and artificial neural networks (ANN) have been used to predict the category to which the data belongs.

While smart contracts enhance security on the cloud, the IoT network can deploy security approaches to detect attacks early at the edge (device level). These approaches can be categorised into signature-based and anomaly-based IDS. A signature-based IDS can only detect known attacks by comparing input patterns with the database. To tackle this issue, state-of-the-art IDSs use anomaly-based approaches (Garcia-Teodoro et al. 2009) to detect unknown attacks. Anomaly-based IDSs are more memory efficient since the former classify infected packets by comparing them with known patterns (Khraisat et al. 2019) (stored in an extensive database). In contrast, the latter can produce the same result by using lightweight mathematical models. Additionally, anomaly-based IDS can detect new attacks, while conventional systems cannot. This work focuses on an anomaly-based IDS at the edge. This approach brings security functions to the device, thus increasing the possibility of preventing attacks before they cause global damage to the system. In particular, lightweight ML models are trained and implemented on IoT Gateways to detect network attacks.

Hardware accelerators, such as GPUs (Graphic Processing Units) and FPGAs (Field Programmable Gate Arrays) have a vital role in cybersecurity. For instance, with the rapid growth of databases in a system, an anomaly-based IDS demands up-to-date ML models to deal with attacks; GPUs have been proven as a time-efficient platform for training ML models compared to traditional CPUs (Central Processing Unit). While GPUs excel in general-purpose parallel computations, FPGAs adapt better to a specific task, such as inference, with less energy consumption. In terms of IoT devices, security solutions vary according to different perspectives. Edge devices are limited in computing resources and are susceptible to malware, side-channel and authentication attacks. There are more powerful devices, called Gateways, which are usually located between edge devices and the cloud, to collect data and manage edge devices. Gateways face higher-level attacks such as network, data and privacy attacks. This work accelerates lightweight ML models, including ANN and CNN, on an FPGA-based architecture.

1.2 Literature Review

An IoT network architecture can be diversified by combining multiple layers, depending on the network's goal and environment. For instance, a basic IoT network serving a dedicated application may only require minimum data collect on devices (sensors, actuators, etc.) and a host gateway to collect, process, and display information. In contrast, a fully scalable IoT network for an ecosystem may need multiple clusters of local devices connected to its dedicated Gateway. These Gateways can exchange data with the upper layers, such as the application layer, data management layer, cloud layer, etc. This section discusses the related work in IoT security from the architecture's view.

1.2.1 IoT Security: An Architecture View

With continuous and rapid developments in IoT architecture and technology, IoT security can be developed at four different layers (Prasad, Kapoor, and Singh 2024): application, processing, network, or perception layers to counter threats and vulnerabilities. Each layer faces different challenges due to the variety of security threats (Sun et al. 2024). For instance, since edge and sensor devices at the perception layer are limited in computing resources, they require lightweight protection models rather than complex algorithms (Musaddiq et

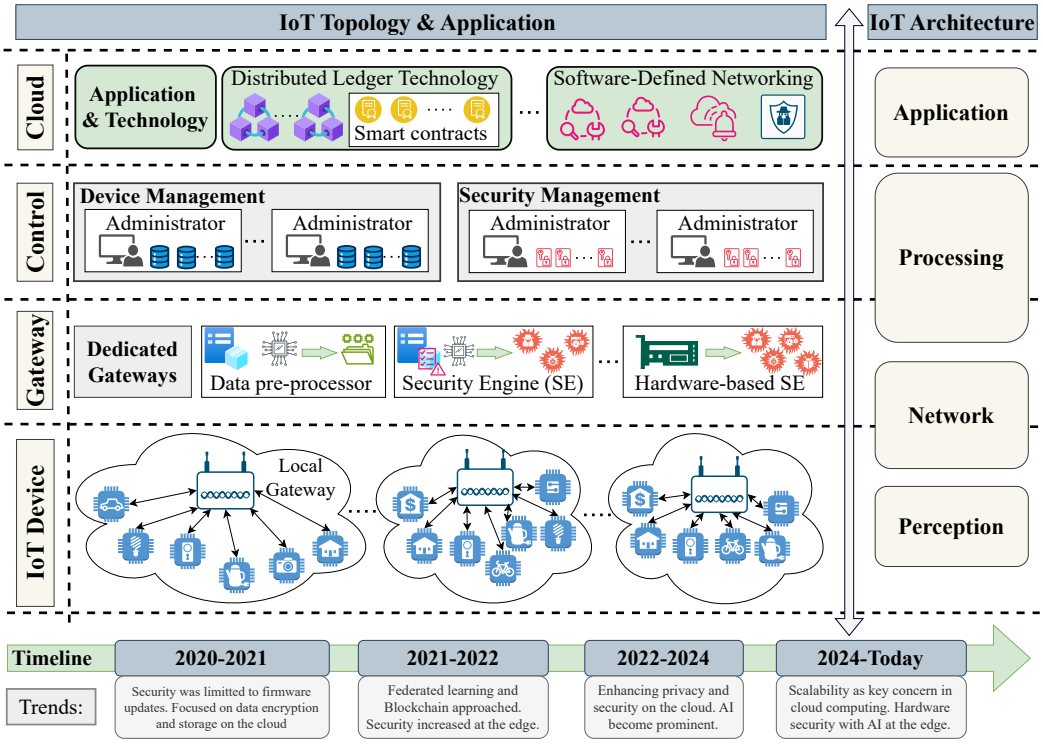


Figure 1.1: A high-level view of an IoT architecture with four layers: application, processing, network, and perception. This architecture is an abstraction of the practical IoT topology and application architecture on the left-hand side.

al. 2022). The network presents the communication channel between devices and administrators; thus, this layer attracts most attacks, such as DoS/DDoS, routing, traffic analysis, spoofing attacks, etc. In the processing and application layer, there are issues related to access control and data processing privacy that need to be resolved. The traditional method with static access control is becoming inefficient in modern systems (S. Shen et al. 2023).

Figure 1.1 illustrates the high-level view of a practical IoT network, including the IoT topology and IoT architecture side by side. This topology has multiple layers, namely the Cloud, Control, Gateway, and IoT Device; they are mapped to the four-layer IoT architecture on the right. This figure visualises the thesis goals, which study scalable security approaches in IoT networks. Blockchain is proposed to enhance scalable security in the cloud while anomaly-based IDSs are implemented across the control and device layers. Specifically, lightweight ML models are used at the hardware-based Gateway level for early attack detection in high-speed network environments. This section summarises a high-level view of existing works in IoT Security from the edge to the cloud utilising Blockchain, anomaly-based IDSs, and hardware accelerations.

IoT Security trends have evolved toward integrating AI, Blockchain, and decentralized solutions, emphasising scalability and security (Rupanetti and Kaabouch 2024). Each layer of IoT architecture has seen specific advancements, and the focus has shifted from traditional security methodologies to highly adaptive, scalable and decentralised solutions (Jewani and Abimannan 2023).

In addition, state-of-the-art IoT security systems must emphasize scalability as the number of new devices in the network continues to grow exponentially with the integration of sophisticated technologies. Three critical aspects of an IoT security system must be addressed: system management and scaling, the integration of new AI techniques for security, and adaptation to high-speed network environments.

To tackle these challenges, the thesis focuses on scalable security approaches for IoT networks utilising distributed ledger technology, anomaly-based intrusion detection systems (IDSs), and hardware accelerators. The following sections present the related work to support the proposed approaches.

1.2.2 IoT Security on The Cloud: Smart Contract Technology for IoT Access Control

Blockchain and smart contracts have had a big impact on IoT security lately. It has been involved in artificial intelligence of things (M. Shen et al. 2023) as a solution in terms of efficiency, security, privacy, trust and incentives. The authors in (Yazdinejad et al. 2022) have proposed Blockchain-enhancing federated learning in a cluster-based architecture for anomaly detection in IIoT networks. According to researchers in (Shammar, Zahary, and Al-Shargabi 2021) and (Reyna et al. 2018), Blockchain and smart contracts technology enhances IoT security in different aspects, including authentication, access control, privacy, trust, and reputation.

Blockchain and smart contracts are widely used in IoT access control and security due to their decentralised nature and data integrity. Table 1.1 summarises existing research results on IoT security using Blockchain.

Table 1.1: Summary of existing literature on IoT security and access control using Blockchain and smart contracts.

Work	Methodology	Results	Network	Categorise
(Algarni et al. 2021)	Proposes a multi-agent system to provide lightweight, decentralised IoT access control security.	Enables secure communication between IoT devices, fog nodes and cloud computing.	Private Blockchain	IoT access control
(P. Kumar, R. Kumar, et al. 2021)	PCA (Principal Component Analysis) technique was used to transform raw data, which was then transmitted by Blockchain. Gradient Boosting Anomaly Detector (GBAD) was trained on ToN-IoT and BoT-IoT datasets to evaluate the framework.	Achieves improved efficiency since it is immune to inference and data poisoning attacks.	Ethereum Blockchain	IoT privacy and security
(Wickström, Westerlund, and Pulkkis 2021)	An authentication model for IoT devices and the distribution of tasks using two types of smart contracts: manager and device contracts.	Smart contracts were tested and verified with reasonable transaction costs. The system can handle authentication and authorisation.	Ethereum's Ropsten testnet	IoT access control
(Abubakar et al. 2022)	Uses smart contracts on the Ethereum network as an additional method in the two-factor authentication (2FA) process for web-based access to sensor data.	Smart contract functions are tested on a test net.	Ethereum's Ropsten testnet	IoT authentication

(Dwivedi, Amin, and Vollala 2023)	A Blockchain-based framework for data authentication and storage in an IoT environment. The design includes three layers: IoT devices, fog nodes and IPFS. Smart contracts are deployed at the fog nodes to authenticate and transfer data to the storage located at the IPFS layer.	Only authorised users can access the data stored on the off-chain storage system. The cost analysis confirms a linear relationship between the transaction and execution cost and transaction fee and transaction cost.	Ethereum's Ropsten testnet	IoT privacy and authentication
(H. Shah et al. 2023)	Proposes a two-layer security approach for an IoT environment. The first layer filters legitimate users by using ML techniques to filter network packets sent by users. Legitimate users are granted permission to deploy smart contracts. These contracts are fed into ML models, which present the second layer of security.	Detects the intruders in the IoT system with an accuracy of 99.72% and predicts whether the smart contracts are malicious or non-malicious, with an accuracy of 99.27%.	Ethereum Blockchain	IoT and smart contract security
(Bobde et al. 2024)	A Blockchain gateway classifies data based on probability levels and decides their storage in cloud servers or the Interplanetary File System. This system employs Zero Knowledge Proof for device authorisation and balances security and accessibility.	Blockchain protects the data from sensor nodes and sends it to a Blockchain gateway. Both Zero Knowledge Proof and federated learning enhance framework security.	Private Blockchain (PoA)	IoT security and access control

(Zhonghua, Goyal, and Rajawat 2024)	Proposes three levels of access control architecture with a smart contract-based attribute-based access control model in the middle. It comprises four smart contracts, namely management, judge, punishment and access control contracts.	The system passed the security analysis. The Blockchain test result shows that when the number of accesses increases, the time spent on each access increases linearly.	Private Blockchain (improved PoW)	IoT access control
(Pittaras et al. 2024)	A novel design of the smart contract-based digital twins on two Blockchain scenarios (public and private). Introduced an architecture for accessing and controlling IoT devices using Blockchain.	Securing the real IoT devices and the gateways by allowing consumers to interact with them only through their smart contract-based digital twins.	Ethereum and Hyperledger Fabric Blockchain	IoT access control
(Rashid, Masood, and A. u. R. Khan 2024)	Proposes smart contracts for registering devices and creating policies in IoT networks.	Gas costs and latency were measured. The policy size greatly affects gas units consumed.	Ethereum testnet	IoT access control

In (Abubakar et al. 2022), the authors proposed Blockchain for two-factor authentication. The authentication is deployed at the application level, and there are no countermeasures if the devices are infected. The authors in (Algarni et al. 2021; Wickström, Westerlund, and Pulkkis 2021; Zhonghua, Goyal, and Rajawat 2024) and (Rashid, Masood, and A. u. R. Khan 2024) have focused on high-level IoT access control using smart contracts. The designs are purely based on smart contracts on the Blockchain. The research (Zhonghua, Goyal, and Rajawat 2024) has combined smart contracts in Blockchain with attribute-based access control. However, the work only focuses on access control under the edge Internet of Things and does not deal with the existing environment.

Similarly, Blockchain has been used as an intermediate mechanism for consumers to interact with IoT networks (Pittaras et al. 2024). The gateways, which control edge devices, can interact with smart contracts. In (Dwivedi, Amin, and Vollala 2023) and (Bobde et al. 2024), the authors have proposed

off-chain solutions for data storage since it is expensive to store data on a live Blockchain (or on-chain). In particular, the authors in (Bobde et al. 2024) have used a private Blockchain with PoA consensus to deploy security from the sensor nodes' level. However, the work did not test the system's scalability. In (P. Kumar, R. Kumar, et al. 2021) and (H. Shah et al. 2023), the authors have combined smart contracts and IDS for IoT security and access control. This concept has added an extra layer of security to IoT networks. In addition, security in a smart contract is also a concern (H. Shah et al. 2023) and requires careful attention to build a comprehensive secured system. While much research focuses on designing smart contracts, IDS and hardware accelerators can potentially add efficiency and scalable security to IoT networks.

1.2.3 IoT Security at The Edge: Anomaly-based IDSs Using Hardware Accelerator

IDSs (Intrusion Detection Systems) can be categorised into two groups: signature-based and anomaly-based IDSs, depending on the detection techniques used in the system. This section discusses the anomaly-based IDSs using ML to detect network attacks.

Machine learning and deep learning (DL) have been widely adopted in anomaly-based IDS to detect unseen attacks (Yang et al. 2022; Alsoufi et al. 2021). The authors in (Hasan et al. 2019; P. Kumar, G. P. Gupta, and Tripathi 2021) have experimented with various machine learning algorithms on an IoT-based open-source dataset. They have achieved up to 99% accuracy by applying the random forest (RF) classifier; however, the dataset used in this research was small (357,952 records) compared to other practical datasets. RF has also been claimed, in (Thamaraiselvi and Mary 2020), as the best algorithm for detecting anomalies. Meanwhile, DL has shown better performance when dealing with large datasets. For instance, researchers in (Vinayakumar et al. 2019) have trained deep neural network (DNN) models on six different datasets with the best training accuracy in the range 95% to 99% on the KDDCup-99 and NSL-KDD datasets. On the same datasets, the system in (Xu et al. 2018) has achieved similar results (up to 99% detection rate) by using a DNN model with an automatic feature extraction method. Even though ML and DL are prominent techniques for anomaly-based IDS with high detection rates, the trained DL models usually require substantial computational resources, which are limited on standard IoT devices.

Table 1.2: Summary of anomaly-based IDS for IoT networks

Work	Dataset	Platform	Accuracy
(T. D. Nguyen et al. 2019)	Self-generated	Redeon Rx 460 GPU	95.6%
(Mothukuri et al. 2021)	Modbus network	Lambda GPU	99.5%
(Vaccari et al. 2020)	MQTTset	Core Intel i7 dual, 16GB of RAM	98.0%
(Manimurugan et al. 2020)	CICIDS 2017	Core Intel i7, 16GB of RAM	99.4%
(Yin et al. 2020)	Yahoo Webscope S5	Google Colab	99.6%
(Bovenzi et al. 2020)	Bot-IoT	Not mentioned	99.0%
(Protogerou et al. 2021)	Self-generated	Not mentioned	97.0%

Table 1.2 shows a summary of anomaly-based IDS for IoT networks. The paper (T. D. Nguyen et al. 2019) has introduced a federated self-learning anomaly detection in IoT networks on a self-generated dataset. Similarly, authors in (Mothukuri et al. 2021) have proposed a decentralised federated-learning approach with an ensembler, which combines long short-term memory (LSTM) and gated recurrent units to enable anomaly detection. Researchers in (Vaccari et al. 2020) have tested various ML algorithms on a generated dataset, called MQTT, for IoT network attack detection and achieved an accuracy of 98%. DL algorithms have also been used to generate detection models from various IoT datasets. For instance, authors in (Manimurugan et al. 2020) have built deep belief network models from the CICIDS 2017 dataset to classify normal records from six different attack types, with an average accuracy of 97.46%. In (Yin et al. 2020), the Yahoo webscope s5 dataset was used as input for convolutional neural network (CNN) and recurrent autoencoder algorithms. Authors in (Bovenzi et al. 2020) have generated lightweight detection models based on a deep autoencoder from the Bot-IoT dataset. The best setup has reached 97.61% F1-score; however, the hardware platform for experimenting with the system is not mentioned. The self-generated dataset is collected in (Protogerou et al. 2021); the data are fed into a graph neural network, which has produced up to 97% accuracy.

In terms of parallel computing platforms, the authors in (Sudeendra Kumar et al. 2017) have mentioned SoC as a future platform for securing IoT devices. Accelerating semantic caching on FPGA has been used in (d'Orazio and Lallet 2018) and (Lallet, Casseau, d'Orazio, et al. 2020) for IoT security monitoring. The work in (Samir et al. 2019) compares ASIC and FPGA for IoT security based on studies from the CAESAR competition (CAESAR 2019). ANN for intrusion detection systems is proposed on an SoC platform in (Ioannou

and Fahmy 2019), they have achieved a speed-up of 161.7 times faster than software-based methods. However, this research has experimented on a labelled NSL-KDD dataset, which is different from classifying network packets. In summary, most of the work in this area has focused on encryption and decryption algorithms instead of machine learning techniques on network packets for detecting attacks.

FPGAs have advantages in parallel processing and reprogrammable capabilities, making them faster and more efficient in executing heavyweight processes, such as image processing, than microcontrollers (Elnawawy et al. 2019). Lightweight ANN models have been trained on the NSL-KDD dataset and deployed on an SoC FPGA platform (Ioannou and Fahmy 2019). As mentioned above, they have reported a speedup of 161.7 times faster than software-based methods. Similarly, (T. Zhang, Tehranipoor, and Farahmandi 2023) has proposed a framework integrating power side-channel capturing and ML models for detecting attacks on a Xilinx ZCU104 FPGA. Specifically, the framework involves data collection and pre-processing, followed by training neural network models using a GPU. The trained model is then quantised and passed through the HLS tool to generate an RTL design, which is deployed onto the FPGA for inference with overhead at the picosecond level.

DDoS attack mitigation for IoT devices using a SoC platform has been investigated in (Brasilino and Swany 2019). This work has implemented a method for defending against DDoS attacks on a ZedBoard (Avnet 2024). In this system, the FPGA assists the ARM CPUs in mitigating DDoS attacks. Experimental results have demonstrated that the system can handle up to 10,000 requests per second before the CPU becomes saturated. This rate is approximately 4.58 Mbps in processing throughput. In (Y. Zhao et al. 2021), the researchers have proposed a design that combines an FPGA and CPU to reduce threat traffic. The FPGA is designed in a pipelined fashion for user instruction analysis. They have reported a maximum forwarding processing capacity of 1.652 Gbps.

From the IoT network point of view, anomaly-based IDSs have to be carefully designed and implemented due to the device's limitations. On the one hand, IoT devices come from different manufacturers, requiring additional hardware configurations and connections. In other words, there has yet to be a consensus regarding protocols and standards between IoT devices; as a result, deploying and establishing communications for these devices is time-consuming. On the other hand, many devices need continuous power to function correctly;

integrating ML applications into IoT devices has to consider both power consumption and computational capability. Additionally, anomaly-based IDS run real-time applications that heavily depend on hardware platforms. Thus, it is important to choose suitable hardware for prototyping and deploying IoT applications.

1.3 Aims of Thesis

The thesis studies scalable security approaches for IoT networks utilising ML (Machine Learning) and DLT (Distributed-ledger Technology) on hardware accelerators. The thesis aims to achieve the following milestones:

- The thesis studies IoT architecture and protocols, focusing on security aspects that enhance scalability and integrability.
- The thesis explores the role of hardware acceleration platforms, including FPGA, GPU, and AI-assisted microcontrollers, in an anomaly-based IDS (Intrusion Detection System) using machine learning algorithms.
- The thesis studies DLT (Distributed-ledger Technology), Blockchain platforms, and Smart Contracts' applications in IoT security.
- The thesis studies the combination of Blockchain and anomaly-based IDSs for scalable security in IoT networks. On the one hand, Blockchain enhances IoT security and scalability in the cloud, while anomaly-based IDSs can detect attacks early at the edge. On the other hand, hardware accelerators are utilised for edge devices in the high-speed network environment.

Finally, the thesis proposes a scalable security framework, which is designed, implemented, and evaluated in a practical network environment. The implemented prototype is evaluated and compared to related works. More importantly, based on experimental results, the thesis concludes with contributions and future directions to facilitate further research.

1.4 Thesis Outline

The thesis is organised into seven chapters, with the remaining chapters being organised as follows:

Chapter 2: *Intrusion Detection System*. This chapter presents research on IDSs,

including background knowledge on network architecture, communication models, and known network attacks. Besides, the chapter studies practical datasets, which will be used to test the proposed approach. Next, an efficient anomaly-based IDS framework for IoT gateways and edge devices using lightweight ANN models is proposed. More specifically, the ANN models are tested with a hierarchical decision-making approach. The experiment employs the IoT-23 dataset (Garcia, Parmisano, and Erquiaga 2020a) for evaluating detection accuracy. The chapter's contribution was published in the International Conference on Field-Programmable Logic and Applications (Ngo, Temko, et al. 2021).

Chapter 3: *Leveraging Hardware Accelerators for Enhancing Security in IoT Networks*. This chapter studies IoT security using IDSs by exploring the framework from the previous chapter. Chapter 3 focuses on hardware acceleration solutions to enhance IoT security at the edge. Firstly, the chapter studies an overview of IoT architecture and protocols to understand the current structure of IoT networks. Following this, the chapter investigates common attacks in IoT networks, focusing on attacks in networks and protocols. Secondly, different computing platforms, including microcontrollers, CPUs, GPUs, FPGAs, and ASICs, are compared and discussed. Specifically, the hardware platforms used for the design, implementation, and evaluation of the proposed approach are highlighted. Finally, the chapter proposes an anomaly-based IDS framework on heterogeneous hardware for IoT devices. The framework aims to generate high-efficiency and low-power consumption ANN models for deploying on hardware accelerators. The chapter's contribution was published in Future Internet journal, MDPI (Ngo, Lightbody, Temko, Pham-Quoc, et al. 2023a).

Chapter 4: *Deep Packet Inspection: Convolutional Neural Networks Approach*. This chapter provides an overview of the history and approaches of deep packet inspection systems. In detail, the proposed CNN models for deep packet inspection are implemented and evaluated. The proposed approach involves labelling network packets in the IoT-23 dataset from pre-trained ANN models; then, CNN models are trained to detect attacks at the packet level. The short-term development plan that optimises the CNN models for implementation on FPGA hardware, which will be put into practice in Chapter 6, is discussed. The chapter's contribution was published in the International Conference on Intelligence of Things, Springer Nature Switzerland (Ngo, Lightbody, Temko, Pham-Quoc, et al. 2023b).

Chapter 5: *Distributed-ledger Technology: Blockchain and Smart Contracts for IoT Security*. This chapter studies Distributed-ledger Technology (DLT) for IoT security in the cloud. Firstly, the chapter studies the background knowledge on DLT, Blockchain, and smart contracts. These technologies form the backbone of modern decentralised systems, with DLT and Blockchain offering secure and immutable data records across the network. Next, the role of smart contracts in managing and securing IoT devices is explored. The chapter focuses on methodologies to enhance the scalability and accessibility of IoT networks using smart contracts. Finally, the proposed approach that utilises smart contracts on the Blockchain to enhance security and scalability in IoT networks is implemented and evaluated. Smart contracts are employed for IoT authentication and authorisation. The smart contracts can coordinate with anomaly-based IDSs to enhance security for IoT networks from the edge to the cloud. The chapter's contribution was published in the International Conference on Intelligence of Things, Springer Nature Switzerland (Ngo, Lightbody, Temko, Murphy, et al. 2024b).

Chapter 6: *Framework Design and Implementation*. This chapter encompasses the continuous developments from the previous chapters, including the implementation and evaluation of the proposed security approaches from the edge to the cloud. In detail, the implemented network architecture for IoT networks comprises three layers, namely cloud, control, and IoT devices. The proposed approach enhances system security and scalability by leveraging smart contracts technology and ML on hardware accelerators. Secured functions are designed and integrated into each layer. The chapter's contribution was published in IEEE Access journal (Ngo, Lightbody, Temko, Murphy, et al. 2024a).

Chapter 7: *Conclusion and Future Work*. This chapter concludes the thesis by discussing the implemented solutions and experimental results. Additionally, future research directions are discussed to either improve or develop further avenues of research.

References

Abubakar, Mwrwan et al. (2022). "A lightweight and user-centric two-factor authentication mechanism for IoT based on blockchain and smart contract". In: *2022 2nd International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*. IEEE, pp. 91–96.

Algarni, Sultan et al. (2021). "Blockchain-based secured access control in an IoT system". In: *Applied Sciences* 11.4, p. 1772.

Alsoufi, Muaadh A et al. (2021). "Anomaly-based intrusion detection systems in IoT using deep learning: A systematic literature review". In: *Applied Sciences* 11.18, p. 8383.

Antonopoulos, Christos P and Nikolaos S Voros (2017). "A data compression hardware accelerator enabling long-term biosignal monitoring based on ultra-low power IoT platforms". In: *Electronics* 6.3, p. 54.

Avnet (2024). *ZedBoard*. https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/ visited Dec 16, 2024.

Bobde, Yash et al. (2024). "Enhancing Industrial IoT Network Security through Blockchain Integration". In: *Electronics* 13.4, p. 687.

Bovenzi, Giampaolo et al. (2020). "A hierarchical hybrid intrusion detection approach in IoT scenarios". In: *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, pp. 1–7.

Brasilino, Lucas R. B. and Martin Swany (2019). "Mitigating DDoS Flooding Attacks against IoT using Custom Hardware Modules". In: *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pp. 58–64. DOI: 10.1109/IOTSMS48152.2019.8939176.

CAESAR (2019). *Cryptographic competitions*. https://competitions.cr.yp.to/caesar.html visited on Dec 16, 2024.

d'Orazio, Laurent and Julien Lallet (2018). "Semantic caching framework: An FPGA-based application for IoT security monitoring". In: *Open Journal of Internet Of Things (OJIoT)* 4.1, pp. 150–157.

Dwivedi, Sanjeev Kumar, Ruhul Amin, and Satyanarayana Vollala (2023). "Smart contract and IPFS-based trustworthy secure data storage and device authentication scheme in fog computing environment". In: *Peer-to-Peer Networking and Applications* 16.1, pp. 1–21.

Elnawawy, Mohammed et al. (2019). "Role of FPGA in Internet of Things Applications". In: *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pp. 1–6. DOI: 10.1109/ISSPIT47144.2019.9001747.

Expertsystem (2022). *What is Machine Learning? A definition*. https://www.expertsystem.com/machine-learning-definition/ visited on Dec 16, 2024.

Gao, C. et al. (2019). "Real-Time Speech Recognition for IoT Purpose using a Delta Recurrent Neural Network Accelerator". In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5. DOI: 10.1109/ISCAS.2019.8702290.

Garcia, S, A Parmisano, and MJ Erquiaga (2020a). *IoT-23: A labeled dataset with malicious and benign IoT network traffic*.

Garcia-Teodoro, Pedro et al. (2009). "Anomaly-based network intrusion detection: Techniques, systems and challenges". In: *computers & security* 28.1-2, pp. 18–28.

Gary Gensler (2024). *Statement on the Approval of Spot Bitcoin Exchange-Traded Products*. https://www.sec.gov/news/statement/gensler-statement-spot-bitcoin-011023 visited on Dec 16, 2024.

Hasan, Mahmudul et al. (2019). "Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches". In: *Internet of Things* 7, p. 100059.

Ioannou, Lenos and Suhaib A Fahmy (2019). "Network intrusion detection using neural networks on FPGA SoCs". In: *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, pp. 232–238.

Jewani, Kajal and Satheesh Abimannan (2023). "Edge Intelligence in IoT: Architecture and Applications". In: *2023 8th International Conference on Communication and Electronics Systems (ICCES)*. IEEE, pp. 337–345.

Khraisat, Ansam et al. (2019). "Survey of intrusion detection systems: techniques, datasets and challenges". In: *Cybersecurity* 2.1, pp. 1–22.

Kumar, Prabhat, Govind P Gupta, and Rakesh Tripathi (2021). "Design of anomaly-based intrusion detection system using fog computing for IoT network". In: *Automatic Control and Computer Sciences* 55.2, pp. 137–147.

Kumar, Prabhat, Randhir Kumar, et al. (2021). "PPSF: A privacy-preserving and secure framework using blockchain-based machine-learning for IoT-driven smart cities". In: *IEEE Transactions on Network Science and Engineering* 8.3, pp. 2326–2341.

Kumar, Sudeendra et al. (2017). "Security enhancements to system on chip devices for IoT perception layer". In: *2017 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*. IEEE, pp. 151–156.

Lallet, Julien, Emmanuel Casseau, Laurent d'Orazio, et al. (2020). "Mascara (ModulAr semantic caching framework) towards FPGA acceleration for IoT security monitoring". In: *International Workshop on Very Large Internet of Things (VLIoT 2020)*.

Lionel Sujay Vailshery (2024). *Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030*. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ visited on Dec 16, 2024.

Maitra, S. et al. (2019). "Performance Evaluation of IoT Encryption Algorithms: Memory, Timing, and Energy". In: *2019 IEEE Sensors Applications Symposium (SAS)*, pp. 1–6. DOI: 10.1109/SAS.2019.8706017.

Mandy Sidana (2017). *Types of classification algorithms in Machine Learning*. https://medium.com/@Mandysidana/machine-learning-types-of-classification-9497bd4f2e14 visited on Dec 16, 2024.

Manimurugan, S et al. (2020). "Effective attack detection in internet of medical things smart environment using a deep belief neural network". In: *IEEE Access* 8, pp. 77396–77404.

Mothukuri, Viraaji et al. (2021). "Federated-Learning-Based Anomaly Detection for IoT Security Attacks". In: *IEEE Internet of Things Journal* 9.4, pp. 2545–2554.

Musaddiq, Arslan et al. (2022). "Learning-based resource management for low-power and lossy IoT networks". In: *IEEE Internet of Things Journal* 9.17, pp. 16006–16016.

Ngo, Duc-Minh, Dominic Lightbody, Andriy Temko, Colin C Murphy, et al. (2024a). "A Scalable Security Approach in IoT Networks: Smart Contracts and Anomaly-based IDS for Gateways using Hardware Accelerators". In: *IEEE Access*.

– (2024b). "Enhancing Security and Scalability in IoT Networks Applying Blockchain Technology". In: *International Conference on Cooperative Design, Visualization and Engineering*. Springer, pp. 265–277.

Ngo, Duc-Minh, Dominic Lightbody, Andriy Temko, Cuong Pham-Quoc, et al. (2023a). "HH-NIDS: Heterogeneous Hardware-Based Network Intrusion Detection Framework for IoT Security". In: *Future Internet* 15.1, p. 9.

– (2023b). "Network Attack Detection on IoT Devices Using 2D-CNN Models". In: *International Conference on Intelligence of Things*. Springer, pp. 237–247.

Ngo, Duc-Minh, Andriy Temko, et al. (2021). "FPGA Hardware Acceleration Framework for Anomaly-based Intrusion Detection System in IoT". In: *2021*

31st International Conference on Field-Programmable Logic and Applications (FPL). IEEE, pp. 69–75.

Nguyen, Thien Duc et al. (2019). "DÏoT: A federated self-learning anomaly detection system for IoT". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, pp. 756–767.

Pittaras, Iakovos et al. (2024). "Secure smart contract-based digital twins for the Internet of Things". In: *Blockchain: Research and Applications* 5.1, p. 100168.

Prasad, Ajay, Prachi Kapoor, and Thipendra P Singh (2024). "Security Threats in IOT and Their Prevention". In: *Communication Technologies and Security Challenges in IoT: Present and Future*. Springer, pp. 131–146.

Protogerou, Aikaterini et al. (2021). "A graph neural network method for distributed anomaly detection in IoT". In: *Evolving Systems* 12.1, pp. 19–36.

Rashid, Aqsa, Asif Masood, and Atta ur Rehman Khan (2024). "ACS-IoT: Smart Contract and Blockchain Assisted Framework for Access Control Systems in IoT Enterprise Environment". In: *Wireless Personal Communications*, pp. 1–22.

Reyna, Ana et al. (2018). "On blockchain and its integration with IoT. Challenges and opportunities". In: *Future generation computer systems* 88, pp. 173–190.

Rupanetti, Dulana and Naima Kaabouch (2024). "Combining Edge Computing-Assisted Internet of Things Security with Artificial Intelligence: Applications, Challenges, and Opportunities". In: *Applied Sciences* 14.16, p. 7104.

Samir, Nagham et al. (2019). "Asic and FPGA comparative study for IoT lightweight hardware security algorithms". In: *Journal of Circuits, Systems and Computers* 28.12.

Shah, Harshit et al. (2023). "Deep learning-based malicious smart contract and intrusion detection system for IoT environment". In: *Mathematics* 11.2, p. 418.

Shammar, Elham A, Ammar T Zahary, and Asma A Al-Shargabi (2021). "A survey of IoT and blockchain integration: Security perspective". In: *IEEE Access* 9, pp. 156114–156150.

Shen, Meng et al. (2023). "Blockchains for artificial intelligence of things: a comprehensive survey". In: *IEEE Internet of Things Journal* 10.16, pp. 14483–14506.

Shen, Shigen et al. (2023). "Joint differential game and double deep q–networks for suppressing malware spread in industrial internet of things". In: *IEEE Transactions on Information Forensics and Security*.

Sun, Panjun et al. (2024). "A survey of iot privacy security: Architecture, technology, challenges, and trends". In: *IEEE Internet of Things Journal*.

Thamaraiselvi, D and S Mary (2020). "Attack and anomaly detection in iot networks using machine learning". In: *Int. J. Comput. Sci. Mob. Comput* 9.10, pp. 95–103.

Vaccari, Ivan et al. (2020). "MQTTset, a New Dataset for Machine Learning Techniques on MQTT". In: *Sensors* 20.22, p. 6578.

Vinayakumar, Ravi et al. (2019). "Deep learning approach for intelligent intrusion detection system". In: *IEEE Access* 7, pp. 41525–41550.

Wickström, John, Magnus Westerlund, and Göran Pulkkis (2021). "Smart Contract based Distributed IoT Security: A Protocol for Autonomous Device Management". In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 776–781. DOI: 10.1109/CCGrid51090.2021.00094.

Xu, Congyuan et al. (2018). "An intrusion detection system using a deep neural network with gated recurrent units". In: *IEEE Access* 6, pp. 48697–48707.

Yang, Zhen et al. (2022). "A systematic literature review of methods and datasets for anomaly-based network intrusion detection". In: *Computers & Security*, p. 102675.

Yazdinejad, Abbas et al. (2022). "Block Hunter: Federated Learning for Cyber Threat Hunting in Blockchain-Based IIoT Networks". In: *IEEE Transactions on Industrial Informatics* 18.11, pp. 8356–8366. DOI: 10.1109/TII.2022.3168011.

Yin, Chunyong et al. (2020). "Anomaly detection based on convolutional recurrent autoencoder for IoT time series". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.1, pp. 112–122.

Zhang, Tao, Mark Tehranipoor, and Farimah Farahmandi (2023). "TrustGuard: Standalone FPGA-Based Security Monitoring Through Power Side-Channel". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

Zhao, Yuyu et al. (2021). "Secure IoT edge: Threat situation awareness based on network traffic". In: *Computer Networks* 201, p. 108525.

Zhonghua, Chen, SB Goyal, and Anand Singh Rajawat (2024). "Smart contracts attribute-based access control model for security & privacy of IoT system using blockchain and edge computing". In: *The Journal of Supercomputing* 80.2, pp. 1396–1425.

Chapter 2

Intrusion Detection Systems

In recent years, the fourth industrial revolution (or Industry 4.0) has significantly impacted IoT networks' developments. IoT has found many applications, including manufacturing, medical, financial, smart agriculture and automotive applications, just to name a few. As a result, the number of devices interconnected through IoT networks grew significantly. Privacy and security issues have become one of the most critical concerns for the scalability and widespread use of this technology.

This chapter studies background knowledge in network architecture, including TCP/IP and OSI models. Next, the existing network attacks and intrusion detection systems are explored. The UNSW-NB15 and IoT-23 datasets will be used to test the proposed approach. These datasets, which contain real-world network traffic data, are essential for validating ML models in detecting attacks.

Finally, the chapter proposes an efficient anomaly-based IDS framework for IoT gateways and edge devices using NN models. More specifically, the NN models are then tested with a hierarchical decision-making approach. The experiment employs the IoT-23 dataset (Garcia, Parmisano, and Erquiaga 2020a) for evaluating detection accuracy.

2.1 Network Architecture

2.1.1 TCP/IP and OSI Models

The **TCP/IP (Transmission Control Protocol/Internet Protocol) model**, which was implemented by the US Department of Defence in the 1960s, is the

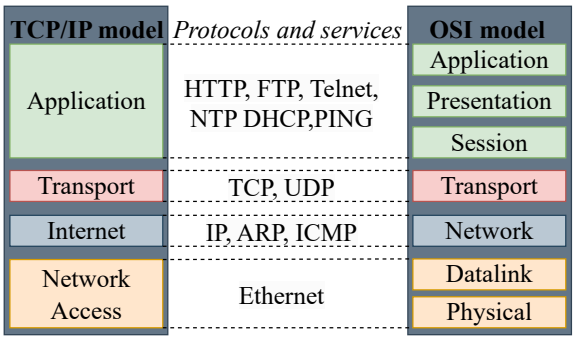


Figure 2.1: The TCP/IP model compared with the OSI model (Fiberbit 2013).

fundamental and most used architecture in digital communication. The structure established connections between points/applications to transmit/receive data correctly. The model uses the protocol to split data into packets instead of sending the whole message; then, each packet can take a different route, depending on the load balancing policy in the routers to which the packet is transmitted. On the receiver side, packets are reassembled to recover the message data.

TCP/IP uses a three-way handshaking mechanism to set up a connection between devices (servers/clients). After a connection is made, the devices start exchanging data through the network, which includes multiple routers, switches, and hubs, to reach other devices. Both the sender and receiver must synchronise and acknowledge packets before communication begins; then, they can transfer TCP socket connections. The TCP/IP model includes four layers, namely Application, Transport, Internet, and Network.

The **OSI (Open Systems Interconnection) model** is another mechanism for transmitting data. Instead of using four layers, the OSI OSI model follows a seven layer structure. They are Application, Presentation, Session, Transport, Network, Datalink, and Physical layers.

Figure 2.1 illustrates the layers' mapping between the TCP/IP and OSI models. The main differences between the two models are in the Application and Network Access layers. The OSI model separates the Presentation and Session from the Application layer in contrast with the TCP/IP model. The Physical layer is not covered by the TCP/IP model since the Datalink layer is designed to be independent of the underlying physical media. This enhances flexibility in the TCP/IP model for adapting to different physical connections, such as Ethernet, Wi-Fi, fibre optics, etc. Overall, both the TCP/IP and OSI models provide a net-

bit offset	0-3	4-7	8-15	16-18	19-31
0	MAC dst (Destination MAC Address)				
⋮	⋮				
112	EtherType				
144	Version	IHL	Type of Service	Total Length	
176	Identification			Flag	Fragment Offset
208	Time to Live		Protocol	Header Checksum	
240	IP src (IPv4 Source Address)				
272	IP dst (IPv4 Destination Address)			IP dst	
304	TCP/UDP src port			TCP/UDP dst port	
336	Sequence Number				
368	ACK (Acknowledgement) Number				
400	Data Offset	Reserved	U A P R S F	Window Size	
432	Checksum			Urgent Pointer	
464	Options (MSS, …)				Padding

Figure 2.2: The packet header structure in the Ethernet II frame standard (Huawei 2024).

working structure for transmitting communication data using a layered system. In both models, each layer has a dedicated purpose, making pinpointing failure points easier.

2.1.2 Packet Structure

Figure 2.2 illustrates the Ethernet II frame and the main fields in the internet and transport layers. Ethernet II frames, also known as Ethernet V2 frames, are the most common Ethernet frames on LANs. Currently, most TCP/IP applications (such as HTTP, FTP, SMTP, and POP3) carry packets over Ethernet II frames (Huawei 2024). Anomaly-based IDS can very effectively learn to detect attacks from this header information.

2.1.3 Network Attacks

According to (Aljabri et al. 2021; Pawar and Anuradha 2015; Packetlabs 2023), network attacks can be classified into two main categories: active and passive attacks.

Active attacks include attacks in which attackers bypass the authorisation process to enter a system, access data, and change it with encrypting or deleting operations. Common active attacks are described below:

- Spoofing attacks are impersonating activities so that the attacker appears

to be a trusted instance. Spoofing attacks can be in different forms, such as email spoofing, IP address, DNS, ARP spoofing, etc.

- Modification attacks happen in forwarding devices (switches and routers), which have routing tables modified by the attackers. The attack causes a delay between devices because packets are travelling using long routes.
- Wormhole attacks also alternate packet routes but use a different mechanism than modification attacks. This can also be called a tunnelling attack, in which an attacker receives a packet at one node and sends it from another malicious node in the network, bypassing normal routing paths.
- Fabrication attacks are attempts to change the routing table by using false routing messages.
- DoS and DDoS (Denial of Service and Distributed Denial of Service, respectively) attacks are flooding attacks in which attackers flood a network or server with excessive traffic to exhaust the computing resources of the target, thus preventing access by legitimate users.
- Sinkhole attacks prevent the server from obtaining correct information. In this attack, a malicious node tries to collect data from neighbour nodes. So the attacker can modify, forward, or drop packets.
- Sybil attack occurs when a malicious entity creates multiple fake identities to manipulate or gain control over a distributed network. In addition, Sybil attacks can happen when a malicious node shares its data with other malicious nodes, increasing the number of malicious nodes in the network. This attack usually happens in distributed systems to gain control over the network.

Passive attacks differ from active attacks since the attackers do not alter the data resources. Instead, attackers illegally access a network and monitor/steal information in the system, including application and user data. Common passive attacks are described below:

- Traffic analysis attacks can reveal insights into the network infrastructure, users' behaviour, and the data content without decrypting it. To achieve this, the attacker first collects information on the communication path between the sender and receiver. Then, it analyses the collected information. There is no modification of data in the traffic analysis attacks.
- Eavesdropping attacks violate privacy in a system. The attacker tries to

gain confidential information by listening to private communications between users. The attacker aims to capture information such as digital keys, credit card details, or any other confidential data being transmitted over the network.

- Monitoring attacks continuously listen to network traffic, system logs, or user activities. The attacker can collect data over time to understand and build up a profile of the system.

2.2 Intrusion Detection System

IDSs (Intrusion Detection Systems) can be categorised into two groups, depending on detection techniques (Azam, Islam, and Huda 2023; Khraisat et al. 2019), which are signature-based and anomaly-based IDS. This section discusses detection techniques used in IDSs, along with existing related work. In particular, state-of-the-art anomaly-based IDSs using ML to detect network attacks are discussed.

2.2.1 Signature-based IDS

Signature-based IDS is designed to detect attacks based on known patterns from a database (Khraisat et al. 2019). For instance, the signature-based IDS monitors activities in a system and triggers alerts if it detects an activity that matches a known attack pattern. The pattern database is built by inspecting the system log files to find sequences of actions previously marked as attacks. By using the exact matching method, signature-based IDS has high accuracy and low false positive rates. However, it can only detect attacks saved in the database; thus, it is ineffective against unknown or zero-day attacks that do not match any known patterns(Tran-Thanh, Pham-Quoc, and Thinh 2018; P. Wang et al. 2016). Snort and NetSTAT are well-known as pioneers in deploying signature-based IDSs.

- Snort is an open-source network intrusion detection and prevention system developed by Cisco. (Roesch et al. 1999) first proposed Snort in 1998. Snort is an efficient real-time traffic analysis tool with versatility and scalability. Besides, Snort is backed by a supportive community that ensures ongoing developments and adoptions. Going beyond an IDS capability, the latest Snort involves intrusion prevention mechanisms (Intrusion Prevention System - IPS) to stop attacks.

- NetSTAT (Vigna and Kemmerer 1999) (Network Security Threat Analysis Tool) focuses on detecting and analyzing network-based threats. While the Snort detection approach is signature-based, NetSTAT uses a state-based approach. This approach looks at the system state and context to generate decisions. NetSTAT offers a comprehensive framework for monitoring network traffic.

Some of the latest cloud systems, including Microsoft Azure, use signature-based IDS (Gustavo Modena 2023) to detect threats. The development process shows efforts to reduce false positive rates while rapidly detecting threats. For instance, Microsoft's Azure Firewall Premium offers advanced protection features with an up-to-date database that contains more than 67,000 signature rules. They also offer fine-tuned controls to enhance the efficiency and flexibility of the system.

Hybrid systems (Rehman, Mushtaq, and Zaman 2024; Li et al. 2024) that combine signature-based methods with anomaly detection are emerging to leverage the strengths of both approaches. These recent trends in signature-based IDS indicate a shift towards integrating advanced technologies to enhance detection capabilities and address limitations inherent in traditional methods. One notable development is the integration of ML techniques (Ahmed et al. 2025) to identify network attacks. For instance, the research (Rahim et al. 2025) enhances a signature-based IDS by integrating a DL classifier to detect attacks. The host-based IDS is designed to be context-aware, using domain-level intelligence and a feedback loop to improve detection. This combination of ML techniques and signature-based IDS provides a more comprehensive attack countermeasure.

2.2.2 Anomaly-based IDS

A state-of-the-art solution for IoT security is an anomaly-based intrusion detection system (IDS). Compared to traditional IDSs, anomaly-based IDSs are more memory efficient since the former classify infected packets by comparing them with known patterns (Khraisat et al. 2019) (stored in an extensive database). In contrast, the latter can produce the same result by using lightweight mathematical models. Additionally, anomaly-based IDS are capable of detecting new attacks, while conventional systems cannot.

An anomaly-based IDS detects attacks using mathematical models of the be-

haviours, which are built from ML, statistical-based or knowledge-based methods. These models measure the deviation in the input behaviour and compare it to a defined threshold. The behaviour is considered an attack/anomaly if the deviation exceeds the threshold. There are three main phases of generating ML models in an anomaly-based IDS, namely training, validation, and inference.

- The training phase is the learning phase, in which the model learns from sample data. Typically, a large portion of the dataset is allocated to use in the training phase compared to the validation and inference phases. The goal is to learn the similarities between the inputs and the outputs.
- The validation phase evaluates the model's performance on a separate subset of the dataset. This phase fine-tuned the model's parameters for the next training epoch. The validation results can also be considered as factors when selecting the best model in the training phase.
- The inference phase is separated from the training and validation phases. The model with the highest accuracy from the validation phase is chosen for deployment in the inference phase. This phase aims to make predictions on new, unseen data.

Overall, anomaly-based IDSs use ML algorithms to build models for detecting attacks. These algorithms are applied to datasets through the training phase to generate ML models. ML models in an anomaly-based IDS can be trained using two main approaches: supervised and unsupervised learning (Maseer et al. 2021). Part of this dissertation focuses on neural network techniques using the supervised learning approach.

2.2.2.1 Supervised Learning Approach

In the supervised learning approach, ML models are trained on labelled datasets. Each input sample is assigned to a specific category (or output label). In the training phase, the model adjusts parameters based on the error function, which calculates the difference between the predicted output and the actual output. The supervised learning approach is used in classification and regression problems (IBM 2024a).

There are numerous machine learning algorithms and their variants used in the supervised learning approach, such as Neural Networks (NN), Naive Bayes (NB), Linear Regression (LR), Support Vector Machines (SVM), K-Nearest Neighbor (KNN), and Random Forest (RF). These algorithms are commonly

used to address most problems. For example, NNs are widely acknowledged in a broad range of applications due to their flexibility in modelling complicated problems. A NN includes layers of nodes that imitate the human neural system by learning from past events. In each neuron, there are parameters (weights and a bias) to generate an output from inputs in the previous layer. NNs learn this mapping function, adjusting parameters based on the loss function.

2.2.2.2 Unsupervised Learning Approach

In the unsupervised learning approach, ML models are trained with unlabelled data. The goal is to discover hidden patterns or data clustering so the trained models can place unseen data in the similarity group. While the supervised learning approach is used in classification, the unsupervised learning approach shows advantages in solving clustering and dimensionality reduction problems (IBM 2024b). The most commonly used ML algorithms in this approach are K-Means Clustering, Hierarchical Clustering, Principal Component Analysis (PCA), and Autoencoders.

2.3 Datasets

There are limitations in the number of available datasets used in IoT for network security. For instance, several recognised datasets in the field of cybersecurity, such as KDDCUP99 (University of California 1999), ISCX (University of New Brunswick 2012) and UNSW-NB15 (ACCS 2018), just to name a few, are frequently used. They are unsuitable for IoT scenarios due to the difference in the protocol and the deployed environment. Table 2.1 discusses common datasets used for training, validating, and inference in anomaly-based IDSs.

Table 2.1: The common datasets in the field.

Dataset	Year	Scenarios	Format	Discussion
IoT-23	2020	This dataset was captured in 23 scenarios from 2018 to 2019 and was published in January 2020. It contains network packets from IoT devices to offer a comprehensive dataset.	The dataset offers pcap files and flow-based summaries data (labelled).	The dataset is well-suited for IoT network environments, with both labelled and raw network data for experimentation.

UNSW-NB15	2015	The dataset is a hybrid of real modern normal activities and synthetic contemporary attack behaviours. This dataset was generated by the IXIA PerfectStorm tool in the Cyber Range Lab of UNSW Canberra.	The dataset offers pcap files and flow-based summaries data (labelled).	The dataset focuses on general network environments. The dataset is widely used in machine learning and deep learning studies to develop advanced intrusion detection systems (IDS) for detecting cyber threats.
KDDCUP99	1999	The dataset contains simulated network traffic with both normal and malicious records, which are categorised into the four attack types: DoS, R2L, U2R, and Probing (port scanning).	There is a total of 41 features extracted from each connection, which are also labelled as normal or attacks. The dataset is split into training and testing sets.	The KDDCUP99 dataset offered feature-rich and labelled records that proved useful in pioneer research in IDSs. However, the dataset has redundant records that need a proper data preprocessing module to balance the data. Not suited for IoT.
NSL-KDD	2009	A refined version of the KDDCUP99 dataset that removes duplicated and unmeaningful records	The content is similar to the KDDCUP99 dataset with an improvement in record distribution.	The dataset is the preprocessed version of the KDDCUP99 dataset, focusing on feature-based methodologies such that pcap files are removed. Not suited for IoT.
ISCX	2012	The dataset was captured in a simulated network, including normal and attack flows (DoS, unauthorised accesses, and HTTP-based attacks).	The dataset offers pcap files and flow-based summaries (the connection between two IP addresses over a period).	ISCX was generated over five consecutive days, allowing IDS systems to consider the data as time series rather than separated instances. Not suited for IoT due to the absence of IoT traffic patterns and protocols.

Bot-IoT	2018	The dataset was generated in a testbed IoT environment (with simulated IoT traffic from multiple devices, sensors, and actuators).	The data is available both in pcap format (raw network packets) and the analysed files that provide forty extracted features for each connection flow.	The Bot-IoT dataset is widely used in IDSs in IoT networks, offering labelled data in both PCAP and CSV formats. Suitable for IoT research in detecting botnets.
MQTT-IoT	2020	The dataset is captured in IoT networks that use the MQTT (Message Queuing Telemetry Transport) protocol. The data focuses on the MQTT protocol data exchange between IoT devices and the MQTT server, including data sharing, command/control messaging, and telemetry data between sensors.	Similar to BoT-IoT, the dataset is also available as pcap and feature files.	The dataset is specialized for MQTT protocol users, making it highly relevant for research on securing IoT networks using this protocol. MQTT traffic can be discontinuous, resulting in difficulty in analysing at the packet level. Suitable for IoT networks using MQTT resource-constrained devices.

Datasets like Bot-IoT and MQTT-IoT are suitable for researching IDSs in IoT environments; these datasets simulate IoT traffic and different protocols needed in a comprehensive network. In contrast, KDDCUP99 and NSL-KDD are primarily focused on legacy networks and are becoming outdated due to limited protocol coverage and attack diversity. ISCX-2012 bridges the mentioned datasets with limited coverage of IoT attacks. This section studies the IoT-23 and UNSW-NB15 datasets, which will be used for evaluating anomaly-based IDSs in this thesis.

2.3.1 IoT-23

The IoT-23 dataset (Garcia, Parmisano, and Erquiaga 2020b) has been created as part of the Avast AIC laboratory. This dataset was captured in 23 scenarios from 2018 to 2019 and was published in January 2020. It contains network packets from IoT devices to offer a comprehensive dataset. The network packets are extracted to produce labelled malware-infected and benign records. Table 2.2 illustrates the total of 325,307,990 labelled records belonging to sixteen categories. Each record has eighteen features, which can be divided into

Table 2.2: The IoT-23 dataset record distribution

#	**Label**	**Number of Records**
1	C&C-Mirai	2
2	PartOfAHorizontalPortScan-Attack	5
3	C&C-HeartBeat-FileDownload	11
4	FileDownload	18
5	C&C-Torii	30
6	C&C-FileDownload	53
7	C&C-HeartBeat-Attack	834
8	C&C-PartOfAHorizontalPortScan	888
9	Attack	9,398
10	C&C	21,995
11	C&C-HeartBeat	33,673
12	DDoS	19,538,713
13	Benign	30,858,735
14	Okiru-Attack	13,609,470
15	Okiru	47,381,241
16	PartOfAHorizontalPortScan	213,852,924

four groups: static, basic, content and time features. Static features can be extracted from the packet header fields. There are five static features in the IoT-23 dataset: IP source, IP destination, source port, destination port and protocol. Other features are calculated from packets in the same flow (or in a time window).

Besides labelled files in the IoT-23 dataset, raw data (network packets) are provided in *pcap* format, which can be replayed to demonstrate the real-time classification process.

2.3.2 UNSW-NB15

The UNSW-NB15 dataset (Moustafa and Slay 2015) is a hybrid of real modern normal activities and synthetic contemporary attack behaviours. This dataset was generated by the IXIA PerfectStorm tool in the Cyber Range Lab of UNSW Canberra. There are 100 GB of raw network packets (pcap files), which are captured using the tcpdump tool. Table 2.3 shows the UNSW-NB15 dataset record distribution for training and inference purposes with a total of 2,540,044 labelled records belonging to ten categories.

Table 2.3: USNW-NB15 dataset record distribution

#	**Label**	**Number of Records**
1	Worms	174
2	Shellcode	1,511
3	Backdoors	2,329
4	Analysis	2,677
5	Reconnaissance	13,987
6	DoS	16,353
7	Fuzzers	24,246
8	Exploits	44,525
9	Generic	215,481
10	Benign	2,218,761

It is widely used in research and practice to evaluate anomaly-based IDSs for cybersecurity. The dataset has different types of attacks: Worms, Shellcodes, Backdoors, Analysis, Reconnaissance, DoS, Fuzzers, Exploits and Generic.

2.4 Artificial Neural Network for Anomaly-based IDSs

This Section tests an anomaly-based IDS using ANN models. More specifically, the training phase is implemented on a GPU to generate NN models, which are then tested using a hierarchical decision-making approach. The experiment employs the IoT-23 dataset (Garcia, Parmisano, and Erquiaga 2020a).

2.4.1 Methodology

2.4.1.1 Artificial Neural Networks

An Artificial Neural Network (Bishop et al. 1995; Jain, Mao, and Mohiuddin 1996) is a computing system that plays an important role in various application domains such as computer vision, speech recognition, or medical diagnosis. ANN imitates the human neural system by learning, recording and using experiences from events. For instance, an ANN computational model is composed of multiple layers, connections, and directions of data propagation. The ANN can learn features of data with multiple levels of abstraction by finding the suitable

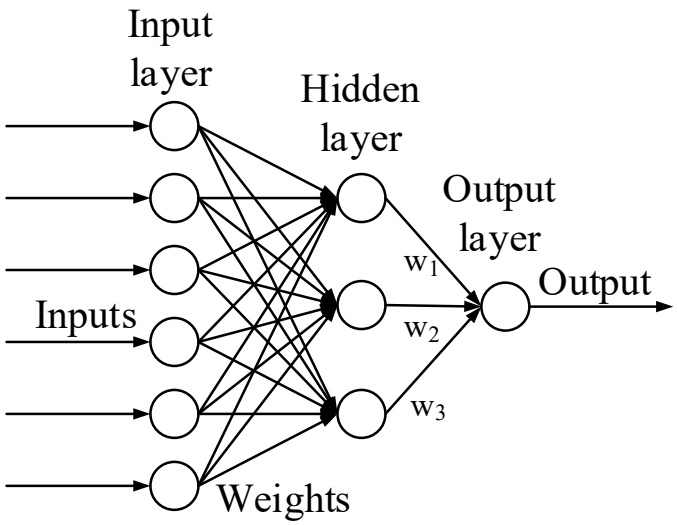


Figure 2.3: A three layers ANN model.

linear or non-linear mathematical manipulation to turn inputs into outputs. Figure 2.3 illustrates an ANN example with three layers.

There are five main elements needed for data processing in ANN, including:

- Inputs are data entries (features).
- Output is the predicted result for the corresponding inputs.
- Weights represent the significance of input features.
- Summation function sums up the weighted input entries for each neuron as shown in Equation 2.1.

$$y = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

  where x_i, w_i and y are the i^{th} input, weight and corresponding result, respectively.

- An activation function is applied to the output of a neuron in an ANN model. This function introduces non-linearity into the model and compresses outputs to a certain range, making it easier for the network to process.

The learning process, referred to as the training phase, of a neural network happens to determine values of parameters or hyper-parameters (such as the number of neurons in hidden layers, weights for activation functions and bias

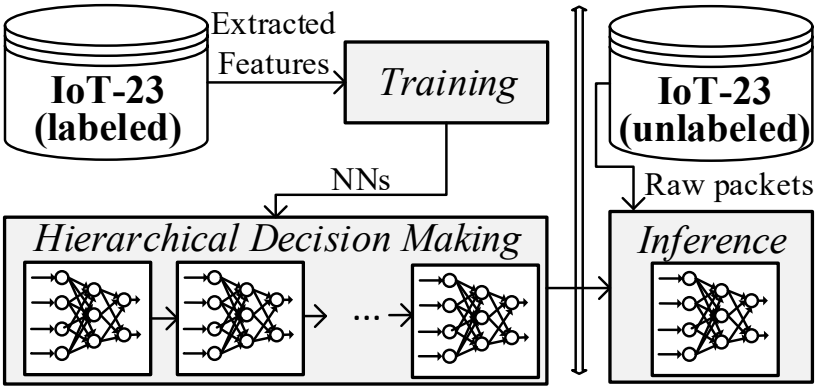


Figure 2.4: System overview in training, hierarchical decision-making, and inference phases using ANN on the IoT-23 dataset.

values) from training datasets. When training a neural network, datasets are fed into the first layer of the network, also known as the input layer. Based on the task being performed, each individual neuron will be assigned a weight value to the input until it memorises the correct output. The trained neural network, then, can be used to perform its tasks by computing the output of the network according to new inputs and assigned weights. This is referred to as the inference phase.

Figure 2.4 illustrates a system overview using NN models on the IoT-23 dataset. In the training phase, the labelled data are used to choose the suitable model architectures and train model parameters (weight and bias values). In the inference stage, the trained model receives input features and produces output values. Further details of the training configuration are presented in Section 2.4.2.3.

The hierarchical decision-making approach in (Sethi and Q. Zhang 2012) is widely used in many areas for reducing problem complexity. This paper uses multiple models, instead of just one, in the inference phase. Applying this method simplifies the model configuration step and improves accuracy, and solves data unbalancing issues. Besides these advantages, this method does have trade-offs by introducing overheads such as increased computational latency and power consumption.

2.4.1.2 Dataset

The IoT-23 dataset, introduced in Section 2.3.1, will be used to build NN models. Table 2.4 shows a total of 325,307,990 labelled records (or flows) belong-

Table 2.4: Sixteen labelled flows in the IoT-23 dataset, divided into five groups

Group	Label	Number of records
1	C&C-Mirai	2
	PartOfAHorizontalPortScan-Attack	5
	C&C-HeartBeat-FileDownload	11
	FileDownload	18
	C&C-Torii	30
	C&C-FileDownload	53
	C&C-HeartBeat-Attack	834
	C&C-PartOfAHorizontalPortScan	888
	Attack	9,398
	C&C	21,995
	C&C-HeartBeat	33,673
2	DDoS	19,538,713
3	Benign	30,858,735
4	Okiru-Attack	13,609,470
	Okiru	47,381,241
5	PartOfAHorizontalPortScan	213,852,924

ing to sixteen categories, including benign class and fifteen combinations of attacks made from nine single attacks.

These labels are separated into five groups in Table 2.4. The first group has attack types with less than 35,000 records each and mainly includes *C&C* infected flows. The second and third groups contain only *DDoS* and *Benign* records, respectively, while the last two groups hold *Okiru* and *PartOfAHorizontalPortScan* marked records.

2.4.1.3 Related Work

The IoT-23 dataset (Garcia, Parmisano, and Erquiaga 2020a) has recently been tested with supervised learning methods. For instance, authors in (Dutta et al. 2020a; Dutta et al. 2020b) have evaluated an ensemble method to combine Deep Neural Networks (DNN) and Long Short-Term Memory (LSTM) for improving accuracy. Additional research (Hussain et al. 2020) has also combined IoT-23 with two other datasets to perform anomaly prediction. They have proposed a universal feature set to detect botnet attacks. Nonetheless, the feature extraction process in this system is tool-based, which might cause dependency and bottleneck-related issues (Storcheus, Rostamizadeh, and Sanjiv Kumar 2015). Malware and botnet analysis on the IoT-23 dataset has been adopted in (Stoian 2020) and (Hegde et al. 2020), respectively. They have

stated similar results among algorithms tested and that Random Forest (RF) has produced the best accuracy. RF has also been selected in (Thamaraiselvi and Mary 2020) as the best algorithm for detecting anomalies. These IoT-23 publications have focused on software-based metrics such as accuracy and F1-score; hardware-based metrics, such as power consumption, performance and network throughput, are not mentioned.

Researchers in (Wielgosz and Karwatowski 2019) have introduced a mapping methodology to bring neural network-based models to hardware. The proposed method has achieved a latency of 210 ns on a Xilinx Zynq UltraScale+ MPSoC FPGA running mapped LSTM models. (Zhong et al. 2024) introduces a novel physical layer digital encryption optical transmission method based on delta-sigma modulation, implemented using FPGAs for real-time transceivers. This system's limitation lies in the trade-off between device compatibility and theoretical security. The absence of true quantum randomness may reduce security, and the memory of the LUT for parameter adjustments is limited. Diwall (El Bouazzati et al. 2025), a lightweight host IDS implemented on an FPGA for detecting wireless attacks targeting IoT devices. Diwall monitors network metrics using hardware performance counters to detect memory corruption, packet injection, and jamming attacks. However, adaptive attackers can evade detection by manipulating HPCs. Besides, it may require retraining due to patches and initially focuses on remote attackers, thus, the threat model could be extended.

This chapter proposes an efficient anomaly-based IDS framework for IoT gateways and edge devices using NN models. More specifically, the ANN models are then tested with a hierarchical decision-making approach. With the goal of accelerating ML algorithms, the thesis targeted FPGA hardware for implementation. ANN models can learn to recognise similar patterns after training processes on labelled datasets. More importantly, an ANN model's inference process is straightforward, and the parameters in the models can be quantised and converted into integers, which aligns well with FPGA hardware architecture.

2.4.2 Proposed Architecture

2.4.2.1 Overview

The Figure 2.5 illustrates the process flow of the proposed anomaly-based IDS framework. There are four phases in the framework: data preparation, training,

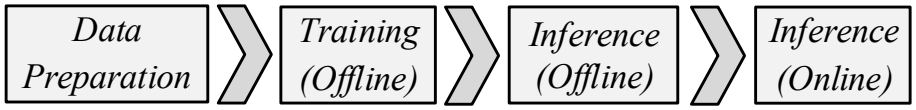


Figure 2.5: The process flow of the proposed anomaly-based IDS framework in four phases: Data preparation, Training (offline) and Inferences (offline and online).

offline inference and online inference. The word "Offline" indicates the static learning process when administrators work with stored data, while "Online" means the action of processing stream data in a real-time context. The purpose of each phase is described as follows:

- *Data preparation*: pre-processes the dataset to appropriate formats used as inputs in the three following phases. This process could be implemented via online or offline approaches; however, the latter is recommended to enhance the framework's training performance.
- *Training*: contains a GPU-based trainer to generate lightweight NN models. This phase is proposed in an offline approach, so as to avoid poisoning attacks in the dataset.
- *Inference (Offline)*: examines the combination of different trained models to gain better accuracy.
- *Inference (Online)*: is an online approach to classify network packets in real time. Furthermore, programmable hardware like SoC, FPGA is proposed to accelerate this process.

The architecture showing each phase's data flow and connections is explained in detail in the following sections.

2.4.2.2 Data preparation

The proposed framework uses a supervised learning neural network approach to classify network attacks. Therefore, the chosen dataset for training this architecture should be labelled. Figure 2.6 shows that there are three main modules in the data preparation phase, which are;

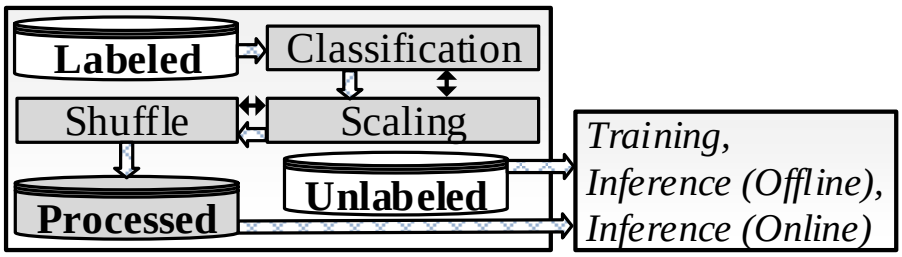


Figure 2.6: The data flow modelling of the Data Preparation phase.

- *The classification* module scans the labelled data, then collects records that have the same label to a new file. The sixteen created files correspond to the benign label and fifteen attacks in the IoT-23 dataset. Then the results are divided into file groups, as illustrated in Table 2.4.

Table 2.5: Inference results (in %) of the selected plans on the IoT-23 dataset

Plan	Indexed group	Input feature set	Output label	Accuracy	Precision	Recall	F1-score	MbA
1	3 and 5	All features	Benign and PHPS	99.75	99.98	99.73	99.86	0.00
2	All 5 groups	All features	Benign and 4 group labels	99.55	99.93	99.57	99.75	0.02
			Benign and 15 attack labels[1]	**99.68**	**99.99**	**99.65**	**99.82**	**0.01**
3	All 5 groups	Static features	Benign and 4 group-attack labels	99.50	99.75	99.69	99.72	0.10
			Benign and 15 attack labels[2]	**99.50**	**99.90**	**99.55**	**99.72**	**0.03**

[1]The best model that covers all 16 types of attacks (model 1). [2]The selected model for testing on FPGA hardware (model 2).

- *The scaling module* transforms extracted features of each record in the alphanumeric form to float values (between 0 and 1) using min-max normalisation.
- *The shuffle* module selects and rearranges records according to different training plans (tuples). In addition, data balancing can be applied in this module for better data distribution. Although the data provided is flow-based information, the framework shuffles these records for the NN to learn static information in the network packets.

The processed data are stored separately and later can be used in other phases. Especially, the framework uses unlabelled data in the inference phase to demonstrate the real-time classification system.

2.4.2.3 Training

The GPU-based training phase is described in Figure 2.7. There are three main modules in this architecture:

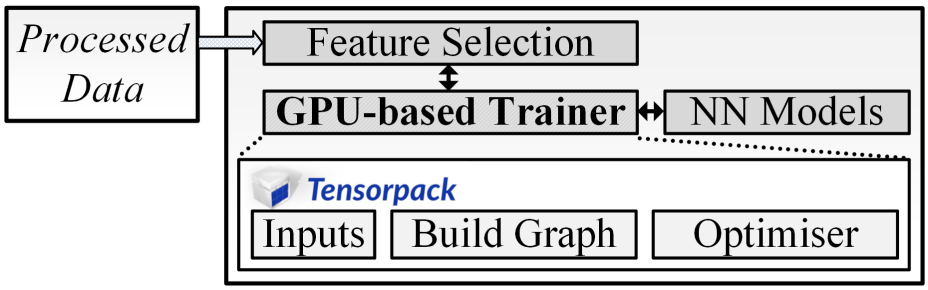


Figure 2.7: GPU-based acceleration for training NN models.

- *The feature selection* module: selects a custom input feature set from each processed record for training.
- *GPU-based Trainer* module: follows the Tensorpack (Wu 2020) structure with three sub-modules: *Inputs* declares input shapes associated with the input feature set, *Build Graph* and *Optimiser* define the model architecture and the selected optimiser, respectively.
- *NN models* module: contains trained models for use in the inference phase.

In NN models, inputs depend on the input feature set, which can contain static features, flow-based features, or both. The hidden layer is configured with 40 neurons, each followed by a hardware efficient ReLU (Rectified Linear Unit) activation function; then, the softmax activation function is used in the output layer. The processed data are distributed at the rate of 50% for training, 25% for validation, and 25% for testing/inference. The choice of 40 neurons balances efficiency and performance for hardware inference. A smaller model ensures real-time processing and meets resource constraints. This number was determined experimentally, with validation accuracy guiding the selection to optimise both accuracy and computational efficiency.

Given that this dataset is applied to network data, and considering the relatively large size of the dataset, allocating 50% of the data for training is reasonable. This ensures that the model has sufficient data for learning, while still retaining an adequate portion for evaluation, allowing for reliable performance assessment. The NN models were trained using the cross-entropy loss function and Adam optimiser with a batch size of 256, and the learning rate is 0.01. The

average training and validation time for 1 epoch is 1,183s and 516s on the Quadro M2000 GPU, respectively.

The NN models are created from three different training plans, as shown in Table 2.5. The first two plans use all the provided features for testing the software-based implementations, while the third plan generates NN models from static features for deploying into FPGA SoC. In this table, the first column contains the training plans, and the second column indicates the selected data from Table 2.4. The third and fourth columns indicate the input feature set and the number of output labels, respectively. The remaining columns represent the inference results in percentages: *Accuracy*, *Precision*, *Recall*, *F1-score* and *MbA*. The test defines the *MbA* metric for measuring the percentage of the number of *Misclassified between Attack* records. According to the inference results, the first training plan recorded the highest accuracy of 99.75%; however, this model can only classify between benign and PartOfAHorizontalPortScan (PHPS). The model 1 is considered the best model, which achieves 99.68% and 99.82% accuracy and F1-score, respectively, with only 0.01% of MbA. The third plan produces models for implementing on FPGA. These models for FPGA have an accuracy of 99.50%, which is close to the model in the second plan.

2.4.2.4 Offline Inference

There are many strategies for building a hierarchical decision-making system. This section has combined two models: model 2 in Table 2.5 (denoted as Model A) and the first plan's model (Model B). While Model A is used for global detection, Model B is better at classifying between Benign and PHPS records. Figure 2.8 illustrates the flowchart of the hierarchical decision-making approach for testing NN models.

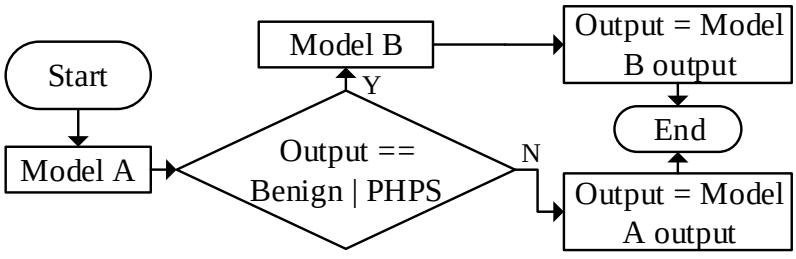


Figure 2.8: Hierarchical decision making flow-chart

The test has compared single NN and multiple NN solutions to related works. Table 2.6 shows systems that have experimented on the IoT-23 dataset. The

authors in (Dutta et al. 2020a) have achieved 99.70% accuracy by using DNN.

Table 2.6: Result comparison to related works in the IoT-23 dataset

System	Model	Accuracy (%)	F1-score (%)
(Dutta et al. 2020a)	DNN	99.70	87.00
(Dutta et al. 2020b)	LSTM	99.99	N/A
(Hussain et al. 2020)	RF	100.00	100.00
(Hegde et al. 2020)	NN	99.80	99.85
(Thamaraiselvi and Mary 2020)	RF	99.50	99.00
Proposed system	NN	99.68	99.82
	Multiple NNs	**99.79**	**99.89**

They have improved this result to 99.99% by applying LSTM in (Dutta et al. 2020b); however, the F1-score is not mentioned in this research. (Hussain et al. 2020) and (Thamaraiselvi and Mary 2020) have reported RF as the best algorithm for detecting attacks with 100% and 99.50% accuracy, respectively. However, the system in (Hussain et al. 2020) has only focused on detecting Mirai records. Authors in (Hegde et al. 2020) have achieved accuracy with only 0.12% greater than the single NN model implementation. By using the hierarchical decision-making solution, the accuracy is close to that in (Hegde et al. 2020), where the authors reported 99.80%. The F1-score from this method is 99.89%, which is the highest result among the related works.

2.5 Summary

This chapter presented some background knowledge in network architecture, including TCP/IP and OSI communication models, network packet structures, and basic categories of network attacks. This knowledge serves as the basis for developing cyber systems.

Following that, signature-based and anomaly-based IDSs are studied to detect network attacks. Signature-based IDS provides high accuracy by matching patterns from a database and raising alerts when an input pattern matches known attack patterns. However, these systems are unable to detect new attacks that have not yet been updated in the database. Anomaly-based IDSs use mathematical models to detect attacks, allowing for the prediction of new attacks.

With the goal of accelerating ML algorithms, the thesis targeted FPGA hardware

for implementation; thus, the ANN model is suitable, as it is lightweight compared to DNN models; at the same time, ANN models can learn to recognise similar patterns after training processes on labelled datasets. More importantly, an ANN model's inference process is straightforward, and the parameters in the models can be quantised and converted into integers, which aligns well with FPGA hardware architecture.

This chapter has presented an anomaly-based IDS using ANN models to detect attacks in IoT networks. The proposed architecture includes dataset preprocessing, GPU-based training acceleration, and offline inferencing using hierarchical decision-making. The proposed algorithms were tested with the IoT-23 dataset using ANN models and achieved the highest accuracy of 99.68%. The accuracy of detection was increased further to 99.79% by applying the hierarchical decision-making method.

The next chapter will focus on detecting network attacks in IoT networks using an anomaly-based IDS. The framework established in this chapter will serve as the milestone for the system discussed in the following chapter. Especially, hardware accelerators are used for the inference phase, as anomaly-based IDSs rely heavily on the computational capabilities of the platform to produce real-time results. Network systems experience varying traffic volumes depending on events, with peak loads or flooding attacks that can result in high-volume traffic; thus, anomaly-based IDS should be capable of handling the high load to mitigate attacks.

References

ACCS (2018). *The UNSW-NB15 Dataset*. https://www.kaggle.com/mrwellsdavid/unsw-nb15 visited on Dec 16, 2024.

Ahmed, Usama et al. (2025). "Signature-based intrusion detection using machine learning and deep learning approaches empowered with fuzzy clustering". In: *Scientific Reports* 15.1, p. 1726.

Aljabri, Malak et al. (2021). "Intelligent techniques for detecting network attacks: review and research directions". In: *Sensors* 21.21, p. 7070.

Azam, Zahedi, Md Motaharul Islam, and Mohammad Nurul Huda (2023). "Comparative analysis of intrusion detection systems and machine learning based model analysis through decision tree". In: *IEEE Access*.

Bishop, Christopher M et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

Dutta, Vibekananda et al. (2020a). "A deep learning ensemble for network anomaly and cyber-attack detection". In: *Sensors* 20.16, p. 4583.

– (2020b). "Detection of Cyberattacks Traces in IoT Data". In: *Journal of Universal Computer Science* 26.11, pp. 1422–1434.

El Bouazzati, Mohamed et al. (2025). "Diwall: A Lightweight Host Intrusion Detection System Against Jamming and Packet Injection Attacks". In: *ACM Transactions on Embedded Computing Systems*.

Fiberbit (2013). *TCP/IP model vs OSI model*. https://fiberbit.com.tw/tcpip-model-vs-osi-model/ visited on Dec 16, 2024.

Garcia, S, A Parmisano, and MJ Erquiaga (2020a). *IoT-23: A labeled dataset with malicious and benign IoT network traffic*.

– (2020b). "IoT-23: A labeled dataset with malicious and benign IoT network traffic". In: *Stratosphere Lab., Praha, Czech Republic, Tech. Rep*.

Gustavo Modena (2023). *Intrusion Detection and Prevention System (IDPS) Based on Signatures*. https://techcommunity.microsoft.com/t5/azure-network-security-blog/intrusion-detection-and-prevention-system-idps-based-on/ba-p/3921330 visited on Dec 16, 2024.

Hegde, Mandira et al. (2020). "Identification of Botnet Activity in IoT Network Traffic Using Machine Learning". In: *2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*. IEEE, pp. 21–27.

Huawei (2024). *Ethernet II Frame*. https://support.huawei.com/enterprise/en/doc/EDOC1100174721/ea0a043c/ethernet-ii-frame visited on Dec 16, 2024.

Hussain, Faisal et al. (2020). "Towards a Universal Features Set for IoT Botnet Attacks Detection". In: *arXiv preprint arXiv:2012.00463*.

IBM (2024a). *What is supervised learning?* https://www.ibm.com/topics/supervised-learning visited on Dec 16, 2024.

– (2024b). *What is unsupervised learning?* https://www.ibm.com/topics/unsupervised-learning visited on Dec 16, 2024.

Jain, Anil K, Jianchang Mao, and K Moidin Mohiuddin (1996). "Artificial neural networks: A tutorial". In: *Computer* 29.3, pp. 31–44.

Khraisat, Ansam et al. (2019). "Survey of intrusion detection systems: techniques, datasets and challenges". In: *Cybersecurity* 2.1, pp. 1–22.

Li, Sifan et al. (2024). "Hda-ids: A hybrid dos attacks intrusion detection system for iot by using semi-supervised cl-gan". In: *Expert Systems with Applications* 238, p. 122198.

Maseer, Ziadoon Kamil et al. (2021). "Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset". In: *IEEE Access* 9, pp. 22351–22370. DOI: 10.1109/ACCESS.2021.3056614.

Moustafa, Nour and Jill Slay (2015). "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 military communications and information systems conference (MilCIS)*. IEEE, pp. 1–6.

Packetlabs (2023). *What Are Network Attacks, and How Can You Protect Against Them?* https://www.packetlabs.net/posts/what-are-network-attacks/ visited on Dec 16, 2024.

Pawar, Mohan V and J Anuradha (2015). "Network security and types of attacks in network". In: *Procedia Computer Science* 48, pp. 503–506.

Rahim, Kashif et al. (2025). "Integrating contextual intelligence with mixture of experts for signature and anomaly-based intrusion detection in CPS security". In: *Neural Computing and Applications*, pp. 1–17.

Rehman, Fazalur, Farhan Mushtaq, and Hafsah Zaman (2024). "A Host-based Intrusion Detection: Using Signature-based and AI-driven Anomaly Detection for Enhanced Cybersecurity". In: *2024 4th International Conference on Digital Futures and Transformative Technologies (ICoDT2)*. IEEE, pp. 1–7.

Roesch, Martin et al. (1999). "Snort: Lightweight intrusion detection for networks." In: *Lisa*. Vol. 99. 1, pp. 229–238.

Sethi, Suresh P and Qing Zhang (2012). *Hierarchical decision making in stochastic manufacturing systems*. Springer Science & Business Media.

Stoian, Nicolas-Alin (2020). "Machine Learning for anomaly detection in IoT networks: Malware analysis on the IoT-23 data set". B.S. thesis. University of Twente.

Storcheus, Dmitry, Afshin Rostamizadeh, and Sanjiv Kumar (2015). "A survey of modern questions and challenges in feature extraction". In: *Feature Extraction: Modern Questions and Challenges*. PMLR, pp. 1–18.

Thamaraiselvi, D and S Mary (2020). "Attack and anomaly detection in iot networks using machine learning". In: *Int. J. Comput. Sci. Mob. Comput* 9.10, pp. 95–103.

Tran-Thanh, Binh, Cuong Pham-Quoc, and Tran Ngoc Thinh (2018). "OpenFlow Switches with Integrated Tiny NIDS to Mitigate Network Attacks". In: *International Journal of Computer Engineering and Information Technology* 10.6, pp. 85–91.

University of California (1999). *KDD Cup 1999 Data*. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html` visited on Dec 16, 2024.

University of New Brunswick (2012). *Intrusion detection evaluation dataset (ISCXIDS2012)*. `https://www.unb.ca/cic/datasets/ids.html` visited on Dec 16, 2024.

Vigna, Giovanni and Richard A Kemmerer (1999). "NetSTAT: A network-based intrusion detection system". In: *Journal of computer security* 7.1, pp. 37–71.

Wang, Ping et al. (2016). "An efficient flow control approach for SDN-based network threat detection and migration using support vector machine". In: *2016 IEEE 13th International Conference on e-Business Engineering (ICEBE)*. IEEE, pp. 56–63.

Wielgosz, Maciej and Michał Karwatowski (2019). "Mapping neural networks to FPGA-based IoT devices for ultra-low latency processing". In: *Sensors* 19.13, p. 2981.

Wu, Yuxin (2020). *Tensorpack Documentation*. `https://tensorpack.readthedocs.io/en/latest/` visited on Dec 16, 2024.

Zhong, Linsheng et al. (2024). "All-digital Device-compatible Real-time Physical Layer Secure Transmission based on Delta-sigma Modulation". In: *Journal of Lightwave Technology*.

Chapter 3

Leveraging Hardware Accelerators for Enhancing Security in IoT Networks

This chapter studies IoT security using IDSs by exploring the framework demonstrated in the previous chapter. This chapter focuses on hardware acceleration solutions to enhance IoT security at the edge (device level). As the number of IoT devices continues to grow, security and privacy become increasingly important. General purpose computing platforms struggle to run detection models in high-speed network environments. Using hardware accelerators is one of the solutions that allow a device to handle higher volumes of traffic with integrated security functions.

As mentioned earlier, anomaly-based IDSs have to be carefully designed and implemented due to the device's limitations. On the one hand, IoT devices come from different manufacturers, requiring additional hardware configurations and connections. In other words, there has yet to be a consensus regarding protocols and standards between IoT devices; as a result, deploying and establishing communications for these devices is time-consuming. On the other hand, many devices need continuous power to function correctly; integrating ML applications into IoT devices has to consider both power consumption and computational capability. Additionally, anomaly-based IDS run real-time applications that heavily depend on hardware platforms. Thus, it is important to choose suitable hardware for prototyping and deploying IoT applications. Hardware accelerators such as GPUs, low-power microprocessors and FPGAs

have a high potential use for IoT devices running ML applications.

This chapter is organised as follows. Firstly, an overview of IoT architecture and protocols is studied to understand the current structure of IoT networks. Following this, an investigation of common attacks in IoT networks, focusing on attacks in networks and protocols, is discussed. Secondly, the chapter studies different computing platforms, including microcontrollers, CPUs, GPUs, FPGAs, and ASICs. Specifically, the hardware that will be used for the design, implementation, and evaluation of the proposed approaches is introduced. Finally, the chapter introduces an anomaly-based IDS framework on heterogeneous hardware for IoT devices. The framework aims to generate high-efficiency and low-power consumption ANN models for deploying on hardware accelerators.

3.1 Background Knowledge

The term Internet of Things was defined in 1999 by Kevin Ashton, a British technology pioneer who co-founded the Auto-ID Center at the Massachusetts Institute of Technology (MIT). The IoT concept has grown from basic device-to-device communication into today's complex, multi-layered architectures with diverse protocols. This development involves technological evolution, research contributions, and industrial demands. Figure 3.1 illustrates the evolution of IoT architecture in four phases.

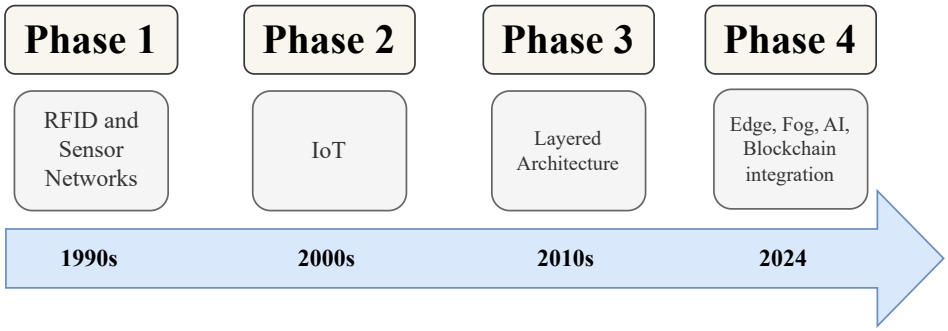


Figure 3.1: The evolution of IoT architecture in four phases.

RFID (Radio Frequency Identification) and sensor networks are the foundation of the interconnected systems, which include devices, networks, and data processing. In the 2000s, the concept of the Internet of Things (IoT) emerged and continues to rise. Researchers (O. Ali et al. 2022; Sachin Kumar, Tiwari, and Zymbler 2019) have focused on expanding connectivity between various

devices and creating digitally connected systems. Next, the 2010s witnessed the introduction of layered IoT architectures, organising IoT systems into layered systems (three and five layers). The current phase integrates disruptive technologies such as Edge computing, Fog computing, AI, and Blockchain into IoT networks. These technologies improve IoT networks by enabling immutable, secured, decentralised data processing and real-time monitoring through Blockchain integration.

As new technologies are integrated into IoT networks, the volume and diversity of data continue to increase, resulting in security challenges. These systems now demand more robust IoT security solutions to safeguard the system against threats. The need for advanced security becomes particularly crucial as IoT networks now carry sensitive data in healthcare, military, and smart cities, where any data breach could have significant consequences. There is a focus on enhanced IoT security protocols, such as Blockchain (Alajlan, Alhumam, and Frikha 2023; Tchagna Kouanou et al. 2022), for securing data and hardware-based solutions to address vulnerabilities in devices with limited computational power. These are also the research directions discussed in this thesis.

3.1.1 IoT Architecture

Similar to network architectures, an IoT architecture can be constructed from multiple layers, with each layer performing specific functions. The number of layers can vary depending on the system's size and design goals. While three-layer and five-layer models are mostly used, there are studies that propose seven-layers (Santos et al. 2020; Krishna et al. 2021) to address open challenges in data processing and security.

Here is an introduction to the three-layer model, which is the fundamental architecture upon which other variants are often built. Figure 3.2 illustrates this model, which consists of the Perception, Transmission, and Application layers, along with their corresponding sub-layers and key technologies (Jing et al. 2014; B. B. Gupta and Quamara 2020):

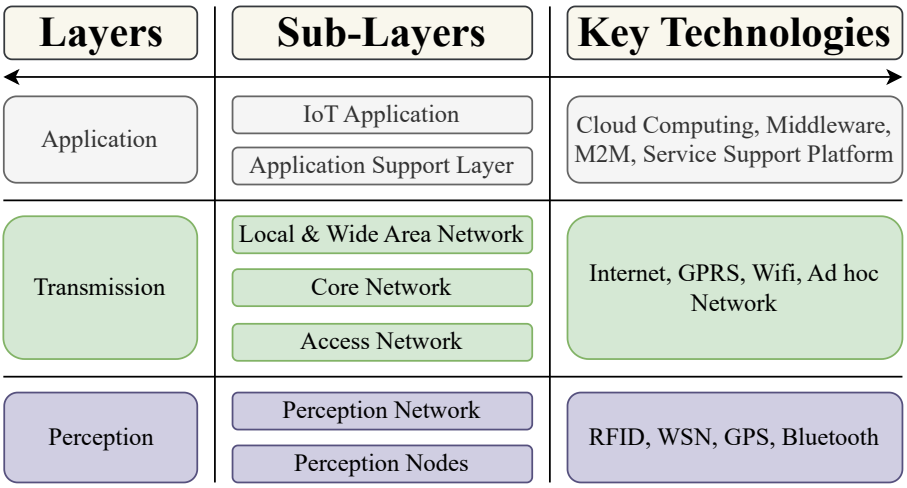


Figure 3.2: The Three-layer model.

1. The perception layer (or physical layer) involves sensors and actuators, which collect data and send it to the transmission layer. For instance, in a smart home system, smart devices (RFID, sensors) are used to capture information such as temperature, humidity, sound, or location. These data can be pre-processed before being sent to the next layer.
2. The transmission layer (or network layer) is responsible for transferring data securely from the perception layer to the application layer. It integrates networking protocols such as Wi-Fi, Ad hoc, etc., to ensure data is transmitted to the destination.
3. The application layer deploys applications for users. This layer analyses and displays the pre-processed data in a user-friendly format.

Researchers (B. B. Gupta and Quamara 2020) have further divided them into sub-layers to describe the core functions constituting each main layer. This approach allows for a better understanding of how each layer operates.

The three-layer model provides a foundational understanding of IoT network architecture. Many variations of this model have been discussed, particularly the five-layer and seven-layer models, each adding more specific functionalities to match the requirements of new IoT applications.

In fact, different organisations propose different numbers of layers, and their specific roles may vary depending on the requirements of the system they are deploying. This flexibility in IoT architecture is due to the wide variety of IoT

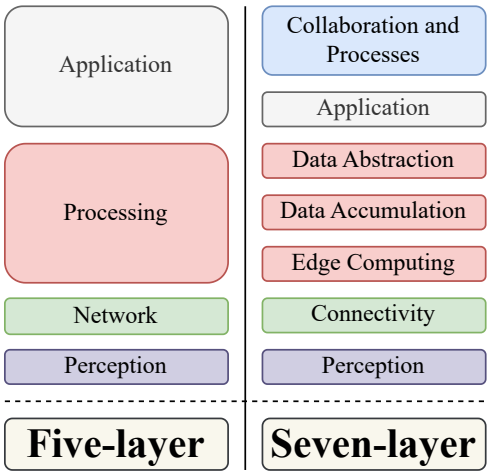


Figure 3.3: The Five-layer and Seven-layer models from Cisco Systems.

applications in sectors like healthcare, manufacturing, smart cities, etc. Each sector demands different levels of data processing and security standards that lead to architectural modifications tailored to specific applications.

For example, Cisco Systems published their own five-layer and seven-layer models (Cisco 2014) in Figure 3.3. The seven-layer model is widely referenced since it addresses different aspects of an IoT architecture, including security, communication, data storage and processing. Edge computing and collaboration layers help the model adapt to state-of-the-art applications (AI) and improve system scalability.

3.1.2 IoT Protocols

IoT protocols ensure data transfer between devices and layers in IoT networks. As the demand for scalability, low power consumption, and security increases, the development process of IoT protocols is continuously improving. IoT protocols can be aligned with the IoT reference model, which means that different layers use different protocols to handle data communications. For the three-layer model, an IoT protocol can be classified into one of three groups (M. A. Khan and Salah 2018) below:

- Application and messaging protocols take care of the communication between applications and provide services to end users. These are high-level communication protocols such as HTTP, MQTT, XML, CoAP, DDS, etc.

- Network and transport protocols transmit data from the perception layer (physical) to the application layer. These protocols handle network data between Gateway devices and the processing centre. For example, IPv6, TCP/UDP, 6LoWPAN, TLS, etc. are popular protocols in this layer.
- Physical devices & communication protocols process data from sensors and edge devices, including wireless and short-range communications. The most used protocols are ZigBee, Bluetooth, WiFi, 4G/5G, LTE, LoRaWAN, NB-IoT, etc.

Overall, each protocol has different functions depending on the data transmitting requirements in each layer. They ensure efficient and secure communications between entities in an IoT network.

3.1.3 Attacks in IoT

Because of the diversity in IoT network configurations (different models and protocols), the attacks targeting IoT networks are also diverse. The previous sections have discussed the reference IoT models and their corresponding protocols. Following a similar format, this section discusses attacks in IoT networks according to the three-layer IoT model.

The Perception layer is where data is collected from sensors and devices. The attacks in this layer usually target hardware vulnerabilities. There are three popular attack types, including Physical, Side-channel, and Spoofing attacks.

- Physical attacks violate the device or sensor physically by replacing or damaging the device. This type of attack occurs when the attacker either has direct access to the devices or the devices lack sufficient protection mechanisms from both hardware and software perspectives.
- Side-channel attacks exploit the hardware behaviour of devices by monitoring and analysing power consumption, electromagnetic emissions, and temperature to extract sensitive information from the device.
- Spoofing attacks are impersonating activities so that the attacker appears to be a trusted instance. Spoofing attacks can be in different forms, such as IP, MAC, or GPS spoofing. The attackers aim to gain unauthorised access or manipulate the network data.

The Transmission layer can be targeted by network attacks, which aim to interrupt or steal information in the data communication process, such as:

- DoS/DDoS attacks are flooding attacks in which attackers send excessive hand-shaking network packets (establishing fake connections) to exhaust the computing resources of the target, thus preventing access from legitimate devices.
- Man-in-the-Middle (MitM) attacks aim at systems that have weak or no encryption mechanism. As a result, the attackers can alter communications between IoT devices.
- Routing attacks change routing information within IoT networks by exploiting vulnerabilities in routing protocols.
- Relay attacks capture original data in the network and replay them to impersonate legitimate communication.

The Application layer has to deal with high-level attacks that are mostly related to privacy issues. For example, malware injection, data breaches, and ransomware are common attacks in this layer.

- Malware attacks aim to deploy malicious code into devices or applications, leading to unauthorised accesses for gaining control or stealing data.
- Data breach attacks steal sensitive information during data processing or storage.
- Ransomware attacks can lock or encrypt data on devices or applications. The attackers then demand a ransom to restore the device's operation.

This chapter focuses on anomaly-based IDSs using ML models to detect attacks based on packet data, which primarily targets attacks in the transmission and application layer in the IoT architecture. Additionally, hardware accelerators are explored to deploy these ML models directly on IoT devices for early detection.

3.2 Hardware Accelerators

Building anomaly-based IDS is a time-consuming process and the inference can be computationally expensive. Hardware accelerators have become a potential upgrade for devices in both edge and cloud-based environments (Tudosa et al. 2019; Microsoft 2021; Liang, Shenoy, and Irwin 2020). Machine Learning (ML) techniques have been widely used to implement various intelligent applications such as speech-recognition (Gao et al. 2019), encryption (Maitra et al. 2019),

biosignal monitoring (Antonopoulos and Voros 2017), etc. These real-time applications require not only high-speed but also low-power hardware to operate sustainably.

This section first studies hardware platforms for AI applications; then, an introduction of specific hardware devices that will be used in this thesis to evaluate the proposed designs and implementations is presented.

3.2.1 Platform Comparison

In recent years, with the explosion of decentralised applications, artificial intelligence, and machine learning, there has been a huge demand for cloud computing services. This phenomenon has prompted cloud service providers to adapt their infrastructure to meet computationally intensive requirements. For example, Microsoft has integrated FPGAs into Azure servers over the past decade, creating a reconfigurable cloud system that can support various application sets. They have stated that: "This reconfigurable cloud provides more efficient execution than CPUs for many scenarios without the inflexibility of fixed-function ASICs at scale". Microsoft has also utilised FPGAs for Bing search ranking, DNN evaluation, and SDN acceleration. Azure's FPGA-based accelerated networking has reduced inter-virtual machine latency by up to 10x, while freeing up CPUs for other tasks (Mark Russinovich 2017).

These advancements in AI and cloud computing have reshaped the roles of CPUs, GPUs, FPGAs, and ASICs. While CPUs remain the most flexible, they are increasingly offloaded for AI/ML workloads due to inefficiencies in parallel processing and power consumption. GPUs continue to dominate AI/ML applications but face growing competition from AI-specific ASICs such as Google's TPUs (Google 2024) and custom AI accelerators. FPGAs are now widely adopted in edge AI and real-time processing applications. ASICs, known for their high efficiency in specialized tasks, have expanded beyond traditional applications to power AI-driven workloads, particularly in deep learning and autonomous systems. These hardware platforms are becoming more specialised, with hybrid approaches leveraging their strengths for different computational phases, from cloud-based training to edge inference (Impulse 2025).

This section discusses the flexibility and power efficiency of different hardware platforms, including CPUs, GPUs, FPGAs and ASICs when dealing with applica-

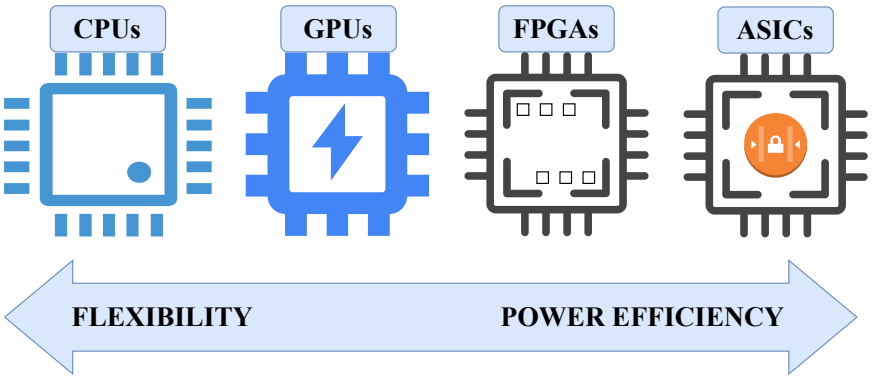


Figure 3.4: The relation between flexibility and power efficiency of CPUs, GPUs, FPGAs and ASICs when dealing with different tasks.

tions. Figure 3.4 compares the four platforms in terms of flexibility and power efficiency for applications.

Overall, CPUs offer the highest flexibility in handling any application. This flexibility is evident in how applications can easily interact with the CPUs computing resources, particularly during the pre-processing and post-processing stages. However, when it comes to the core processing stage, CPUs tend to consume more processing time. Therefore, among the four platforms, CPUs rank the highest in terms of flexibility but the lowest in terms of power efficiency.

In contrast to CPUs, ASICs deliver high power efficiency as each ASIC is manufactured and optimised for a specific application, and it only handles that single type of application. As a result, an ASIC chip lacks the flexibility to deal with other applications but shows undeniable performance and power efficiency when dealing with the application it was specifically designed for.

GPUs and FPGAs are placed between CPUs and ASICs in this comparison. Objectively, GPUs offer superior flexibility compared to FPGAs when handling AI and ML applications, particularly in the performance of training and validating processes. FPGAs have the distinct advantage of being reconfigurable, allowing them to adapt to various applications. However, reconfiguring FPGAs requires engineers to have specialised skills in hardware programming, testing, and verification. These processes are time-consuming. As a result, FPGAs are well-suited for designing and testing applications, serving as a pre-testing stage before producing ASIC chips.

Besides, Microcontrollers (MCUs) are different from the mentioned computing platforms in terms of design goals and types of applications they typically deal

with. They are designed as embedded systems and battery-powered devices that consume minimal power. Microcontrollers have low performance compared to other platforms, but they show an advantage in handling real-time control tasks in IoT devices, automotive control systems, etc.

The thesis uses different hardware platforms to implement the proposed anomaly-based IDSs. For example, CPUs play supporting roles in tasks that require high flexibility, such as data pre-processing and result monitoring. For training and validating ML models, the proposed approach leverages a combination of CPUs and GPUs to achieve higher performance (reducing training time with the same number of training epochs). The inference process using these models is then accelerated on Microcontrollers and FPGA SoCs. Specifically, the targeted hardware platforms include the Max78000 AI-assisted Microcontroller, PYNQ-Z2 FPGA SoC, and Alveo U280 FPGA Card. A brief discussion of these devices is provided below.

3.2.2 Max78000 AI-assisted Microcontroller

Maxim MAX78000 (Maxim Integrated 2020a) has an artificial intelligence microcontroller with ultralow-power neural network acceleration processors. Maxim provides sample projects, which generate NN models using the PyTorch library. The NN processors on the MAX78000 can only support a maximum of 8-bit registers for storing parameters (weights and bias). As a result, the training process is configured as quantisation-aware. The MAX78000 has CNN accelerator cores, which can execute complex AI models for object detection, speech recognition and image classification with extremely low power consumption. Thus, this platform is suitable for edge devices, such as IoT, smart homes, and wearable devices.

3.2.3 PYNQ-Z2 FPGA SoC

The PYNQ-Z2 board (Xilinx 2020) is a versatile FPGA SoC platform for developing hardware-accelerated applications using Python and HDL. This board was developed by Digilent and based on the Xilinx Zynq-7000 SoC device (System on Chip). The platform has an ARM Cortex-A9 processor in the processing system (PS) with an FPGA in the programmable logic (PL) part; this combination allows users to develop and accelerate complex embedded systems and applications. The PYNQ framework enables users to program the FPGA using Python,

making it accessible for users without extensive hardware design experience. The PYNQ-Z2 board can run many applications, such as accelerated NN, signal processing, real-time audio, video, and sensor data processing. Besides, the board is a convenient platform for education and research as it is an accessible platform for learning FPGA programming and embedded systems design.

3.2.4 Alveo U280 FPGA Card

The Alveo U280 FPGA card (AMD 2021) is a high-speed data centre acceleration card developed by AMD Xilinx. The card belongs to the Alveo family, which is designed to accelerate compute-intensive workloads. Furthermore, the card is specialised for high-speed networking applications with fibre-supported ethernet interfaces connected directly to FPGA. Besides, it can accelerate applications in AI, ML, data analytics, and High-Performance Computing (HPC). As part of Xilinx's Alveo portfolio, the U280 leverages the power of the Xilinx FPGA fabric to offer a flexible, reconfigurable hardware platform that delivers low-latency, high-throughput acceleration for demanding applications. The card is equipped with a Xilinx UltraScale+ XCU280 FPGA, which provides 3.8 million logic cells, 902,400 Look-Up Tables (LUTs), and 2,784 DSP slices. The card has high-speed connectivity options, including two QSFP28 ports for 100G Ethernet networking, as well as Gen4 PCIe interfaces for integration into standard server architectures.

3.3 An Anomaly-based IDS Framework using Hardware Accelerators

This Section introduces HH-NIDS - a heterogeneous hardware-based network intrusion detection framework for IoT security. HH-NIDS applies anomaly-based IDS approaches for IoT devices using hardware accelerators. The first prototype system utilises the supervised-learning method on the IoT-23 (Garcia, Parmisano, and Erquiaga 2020b) and UNSW-NB15 (Moustafa and Slay 2015) datasets to generate lightweight ANN models. The proposed framework is implemented on a heterogeneous platform, including a microcontroller, an SoC FPGA and GPU platforms under the handling of host processors. The training phase is conducted on GPU, while the inference phase is accelerated on two hardware platforms:

- The MAX78000 AI microcontroller with ultra-low power consumption (Maxim Integrated 2020a).
- The SoC FPGA is implemented in high-level synthesis (HLS) and Verilog approaches. The Verilog implementation is customised in a pipeline fashion, which has achieved high-throughput and low latency in processing network packets.

Finally, the first prototype system is deployed on the MAX78000 evaluation kit (Maxim Integrated 2020b) (MAX78000EVKIT) and PYNQ-Z2 SoC FPGA board (Xilinx 2020). The implemented systems are evaluated using a number of testing scenarios on the two public datasets: IoT-23 and UNSW-NB15. Hardware resource usage and power consumption are also discussed and compared to relevant works.

3.3.1 Related Works

The IoT-23 dataset has been trained with supervised learning methods. Table 3.1 presents the summary of anomaly-based IDS on the IoT-23 dataset. The authors in (Dutta et al. 2020a; Ullah and Mahmoud 2022) have applied an ensemble strategy, which combines DNN and LSTM. The paper (Hussain et al. 2020) has proposed a universal feature set to detect botnet attacks on the IoT-23 dataset. However, the proposed feature extraction module is tool-based, which can cause dependency and add extra overhead (Storcheus, Rostamizadeh, and Sanjiv Kumar 2015). Malware and botnet analysis on the IoT-23 dataset has been examined in (Stoian 2020; Hegde et al. 2020). Authors in (Nobakht, Javidan, and Pourebrahimi 2022) have reported 99.9% accuracy by applying hyper-parameter optimisation on the deep ensemble model (CNN and RF). Meanwhile, ML algorithms have been used in (Alani and Miri 2022; Douiba et al. 2022) to generate lightweight detection models, resulting in 99.62% and 99.98% accuracy, respectively. However, there are limitations in current works. On the one hand, most of the anomaly-based IDS for IoT networks are designed at the system's high-level view that is usually deployable at the IoT gateway level. Anomaly behaviours can be detected earlier at the edge devices. On the other hand, they have focused on software-based metrics only, while hardware-based metrics, such as power consumption, performance and network throughput, are not mentioned.

The authors in (Alani and Miri 2022) have used a feature selection module to

Table 3.1: Summary of anomaly-based IDS on the IoT-23 dataset

Work	**Dataset**	**Platform**	**Accuracy**
(Dutta et al. 2020a)	IoT-23, LITNET-2020 and NetML-2020	Not mentioned	99.0%
(Ullah and Mahmoud 2022)	NSL-KDD, BoT-IoT, IoT-NI, IoT-23, MQTT and IoT-DS2	Not mentioned	99.8%
(Hussain et al. 2020)	CICIDS2017, CTU-13 and IoT-23	Not mentioned	99.9%
(Stoian 2020)	IoT-23	GTX 1060, 6GB of RAM	99.5%
(Hegde et al. 2020)	IoT-23 and Self-generated	Microsoft Azure	99.8%
(Nobakht, Javidan, and Pourebrahimi 2022)	IoT-23	Intel Core i7-9750H, 32GB of RAM	99.9%
(Alani and Miri 2022)	TON-IoT, IoT-23 and IoT-ID	Not mentioned	99.6%
(Douiba et al. 2022)	NSL-KDD, IoT-23, BoT-IoT and Edge-IIoT	GPU GeForce MX330, 8GB of RAM	99.9%
(Ahli et al. 2023)	IoT-23	Google Colab	98.6%
(Othman and Abdullah 2023)	IoT-23	Not mentioned	99.7
(Nazir et al. 2024)	IoT-23, N-BaIoT, and CICIDS2017	Not mentioned	95.0%
(Al et al. 2024)	IoT-23	Not mentioned	99.2%
(Korba, Diaf, and Ghamri-Doudane 2024)	IoT-23	Google Colab	99.8%

choose the most impactful feature set. Besides, SMOTE (Synthetic Minority Oversampling TEchnique) has been proposed in (Othman and Abdullah 2023) to solve the imbalance issues in the IoT23 dataset and improve the accuracy. Hybrid CNN-LSTM models have been used in (Nazir et al. 2024) and (Al et al. 2024) to achieve 95.0% and 99.2%, respectively, in accuracy. The authors in (Korba, Diaf, and Ghamri-Doudane 2024) have designed a workflow that includes feature extraction & calculation, feature selection, training and testing to reach 99.8% accuracy. On the one hand, most of the mentioned works are software-based, and the research objective is to optimise the accuracy metric. On the other hand, they have not been tested using raw packets to evaluate the proposed ML models in terms of processing throughput. The proposed approach is not only to accelerate detection models using hardware-based platforms but also to be able to process raw packets at high speed.

Hardware accelerators, including specialised AI microcontrollers and SoC FPGA, are promising platforms for securing IoT devices (Sudeendra Kumar et al. 2017; Chéour, Khriji, Kanoun, et al. 2020). IoT security monitoring systems (d'Orazio and Lallet 2018; Lallet, Casseau, d'Orazio, et al. 2020) have implemented accelerating semantic caching on FPGA. Researchers in (Wielgosz and Karwatowski 2019) have introduced a mapping methodology to bring neural network-based models to hardware. The proposed method

has achieved a latency of 210 ns on a Xilinx Zynq UltraScale+ MPSoC FPGA running mapped LSTM models. FPGA acceleration for time series similarity prediction (Kalantar, Z. Zimmerman, and Brisk 2021) has been proposed for IoT edge devices. The implemented system can reach a throughput of 453.5G operations per second with a 10.7x faster inference rate on a Xilinx Ultra96-V2 FPGA compared to the Raspberry Pi. Lightweight ANN models for intrusion detection systems are proposed on an SoC FPGA platform in (Ioannou and Fahmy 2019). They have achieved a speedup of 161.7 times faster than software-based methods on the NSL-KDD dataset. Overall, anomaly-based IDS for IoT devices on hardware accelerators is a promising approach. This chapter introduces an ultra-low power consumption and low-latency heterogeneous hardware-based IDS framework. The prototype system is tested with the IoT-23 and UNSW-NB15 datasets on the MAX78000 microcontroller and SoC FPGA platforms.

3.3.2 Evaluation Metrics

The confusion matrix is widely used to evaluate classification performance. It is a table that shows where the true values belong. Evaluation metrics, including accuracy, precision, recall and F1-score are computed from this confusion matrix. The definition of True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN) for multiple classes can be provided from the confusion matrix (Hossin and Sulaiman 2015).

- Accuracy is one of the popular metrics for evaluating classification models. Equation (3.1) depicts the single-class accuracy measurement.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

- Precision is the positive predictive value. It is a proportion of true positives that the model claims compared to the total number of positives it claims. The single class precision value is given in Equation (3.2):

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

- Recall is defined as the actual positive rate. This number refers to the positives which the model states, compared to the total positives in the

data. The single-class recall value is given in Equation (3.3):

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

- F1-score represents the model's performance. It can be referred to as the weighted average of the precision and recall values. The single class F1-score value is given in Equation (3.4):

$$\text{F1-score} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.4}$$

3.3.3 HH-NIDS Framework

Figure 3.5 illustrates the proposed HH-NIDS framework architecture[1]. The data processing flow is described in three phases, namely *Data Pre-processing*, *Model Generation*, and *Hardware-based Inference*.

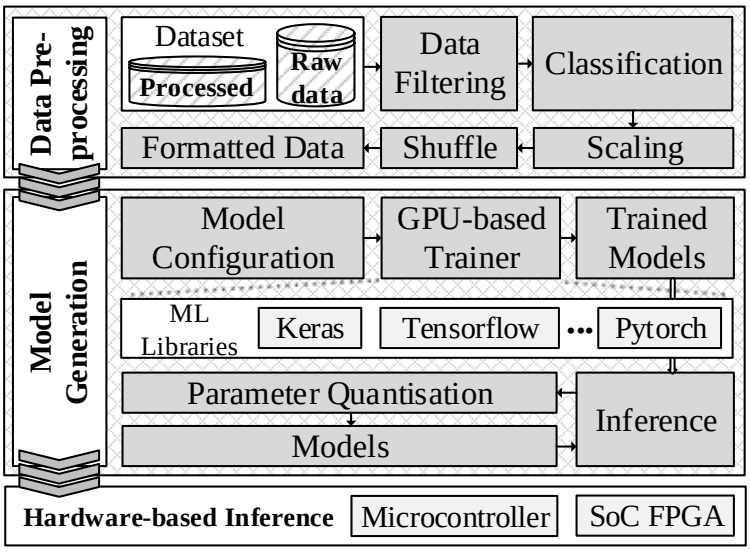


Figure 3.5: The proposed HH-NIDS framework architecture.

Data Pre-processing is the first phase in the HH-NIDS framework to generate data for training and inference processes. HH-NIDS applies a supervised learning neural network approach for detecting network attacks; thus, the chosen dataset should contain labelled records. The raw data (network packets) can

[1]The figure represents the final state of the work and have evolved from the research work and the ongoing iteration.

be used in the inference phase for demonstrating the real-time network classification process. Besides the Dataset block, there are five main functional blocks in this pre-processing phase:

- *The Data Filtering block*: scans the processed dataset (labelled records) and buffers all of the IPv4 records, which make up the majority of the dataset, as input data for the next blocks. In other words, this block creates a custom filter for selecting input records based on the administrators' defined rules.
- *The Classification block*: examines the labelled data, and then collects records that have the same label to a new file. The number of new files is equal to the number of dataset labels/classes (benign and attacks).
- *The Scaling block*: reads the created data files and transforms the record's features, which are initially in alphanumeric or numeric forms, to float values (between 0 and 1) using min-max normalisation.
- *The Shuffle block*: rearranges records according to different training schemes. In addition, data balancing can be applied in this block for better data distribution. Although the network data can be provided with time-based information, this block shuffles these records for the neural network models to learn static information in the network packets.
- *The Formatted Data block*: holds shuffled data files from the previous block; the data are ready to be used in the training and inference processes.

Model generation: this phase includes training, inference and quantisation processes. In the beginning, the training process generates NN models that use GPU-based approaches; then, these models are quantised to fit in accelerator platforms in the hardware-based inference phase. The training process can be described using three blocks:

- *The Model Configuration block*: has two main functionalities, which are: defining training settings and selecting a custom input feature set from each processed record for training. The output data will be forwarded to the *GPU-based Trainer block* for training.
- *The GPU-based Trainer block*: follows the ML library architectures (Keras, Tensorflow, Pytorch, etc.) to train NN models. The HH-NIDS framework proposes GPU-based approaches for better training performance

compared to CPU-based approaches.

- *The Trained Models block*: stores trained models from the *GPU-based Trainer block*. These models will be used in the next inference phase.

Next, the trained models will be evaluated with test data. The administrators can monitor the validation results and select dedicated models for quantisation processes. Figure 3.6 illustrates the HH-NIDS framework's process flow from training to hardware implementations.

- *The Inference block*: represents the software-based inference process. This block uses the trained models on the allocated data files to evaluate how well the trained models react to unseen data. The chosen models then need to be quantised and tested before being deployed into hardware accelerator platforms.
- *The Parameter Quantisation block*: transforms the model's parameters to the supported bit width to fit into the hardware accelerator platforms.
- *The Models block*: contains quantised models, which will be evaluated again by *the Inference block*.

Hardware-based inference: this phase deploys quantised NN models into hardware accelerator platforms for inference. The HH-NIDS framework proposes using two different platforms: namely, a microcontroller and an SoC FPGA. The implemented hardware architecture is discussed in detail in Section 3.3.4.

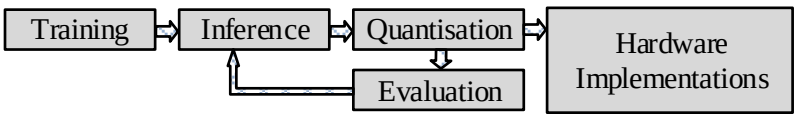


Figure 3.6: The HH-NIDS framework's processing flow from training to hardware implementations.

3.3.4 Implementation

This section introduces HH-NIDS's prototype system implementation. The prototype system uses heterogeneous hardware, including GPU, Microcontroller, SoC FPGA and host processors for experimenting with the IoT-23 and UNSW-NB15 datasets.

Data Pre-Processing

Table 3.2: Flow Features from the IoT-23 and UNSW-NB15 datasets

#	IoT-23	UNSW-NB15	Inputs	Description
1	id_orig.h	srcip	4	Source IP address
2	id_orig.p	sport	1	Source port number
3	id_resp.h	dstip	4	Destination IP address
4	id_resp.p	dsport	1	Destination port number
5	proto	proto	1	Transaction protocol

The HH-NIDS utilises static features for training the NN models. Table 3.2 shows the static features definitions from the two datasets; the source/destination IP address is represented as four 8-bit inputs, while other fields are normalised/scaled to fit into the neurons' inputs.

Model Generation

As described in the previous section, the pre-processing data phase extracts eleven flow features as inputs for NN models. The hidden layer in a NN model is configured with 32 or 48 neurons, each followed by a hardware-efficient ReLU (Rectified Linear Unit) activation function; the softmax activation function is used in the output layer. The number of neurons are chosen to balance efficiency and performance for hardware inference. A smaller model ensures real-time processing and meets resource constraints. This number was determined experimentally, with validation accuracy guiding the selection to optimise both accuracy and computational efficiency.

The processed data are allocated at the rate of 50% for training, 25% for validation, and 25% for inference. The NN models were trained using the cross-entropy loss function and Adam optimiser with a batch size of 512, and the learning rate was set equal to 0.0005, 0.001, or 0.01.

Microcontroller

The inference phase has been deployed on the Maxim MAX78000EVKIT, which has an artificial intelligence microcontroller with ultralow-power neural network acceleration processors. Maxim provides sample projects, which generate NN models using the PyTorch library. The NN processors on the MAX78000EVKIT can only support a maximum of 8-bit registers for storing parameters (weights and bias). As a result, the training process is configured as quantisation-aware.

SoC FPGA

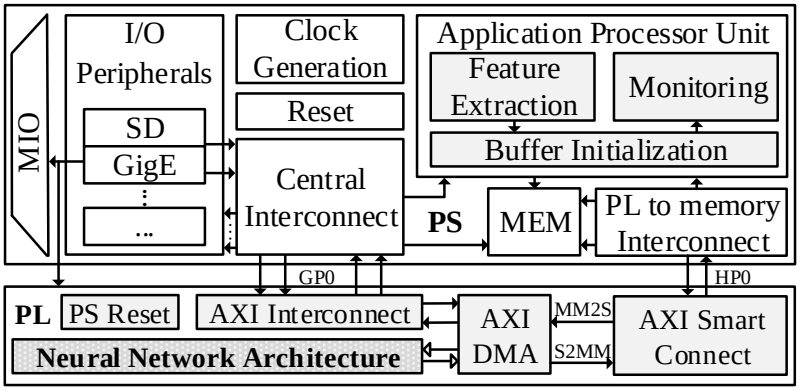


Figure 3.7: Neural network inference acceleration on Zynq-7000 SoC architecture.

Figure 3.7 illustrates the inference phase block diagram on the Zynq-7000 SoC architecture[2], which includes two main domains: namely, the Processing System (PS) and the Programmable Logic (PL/FPGA). The PS has basic modules for a computer system: Multiplexer Input/Output (MIO), I/O Peripherals, Memory (MEM), Interconnects and an Application Processor Unit (APU). The SoC FPGA board is booted from an SD card, and the programmed application is loaded into the APU. Here, network packets will be sent to the APU for extracting input features in the *Feature Extraction* module. Then, the extracted features are grouped together in the *Buffer Initialization* module, before being sent to the PL. This buffer helps reduce overhead in data exchanges between the PS and the PL. Processed results from the PL are delivered back to the *Monitoring* module for analysis. The PL domain has five main blocks, which are:

- *The PS Reset* block: receives the reset control signals from the PS to reset the PL to the initial state.
- *The AXI Interconnect* block: assigns user's input values to registers in PL through General-Purpose Ports (GP0), which is configured as a 32-bit AXI-lite interface. There are two registers to be configured in the *NN* block: namely, the *RESET* register for resetting the module and the *NUM_OF_INPUT* register to indicate the number of records to be sent from PS each time.
- *AXI Smart Connect* block: receives grouped input features, which are

[2]The figure represents the final state of the work and have evolved from the research work and the ongoing iteration.

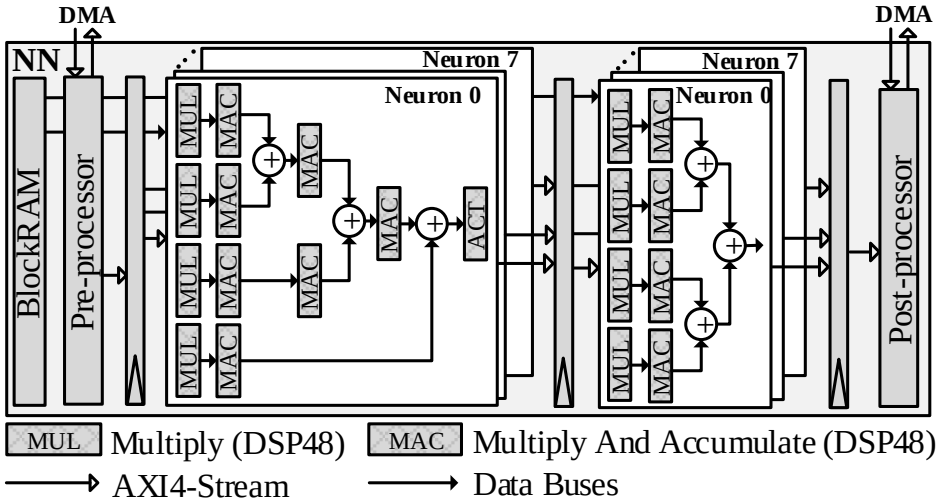


Figure 3.8: The generated FPGA block architecture using the HLS implementation approach.

buffered in the PS, using High-Performance Ports (HP0). Then, data are sent to the *AXI DMA* block by a Memory-Mapped to Stream (MM2S) channel. In addition, the scanned data are transferred back from *AXI DMA* to this block by a Stream to Memory-Mapped (S2MM) channel. These data are intrusion detection results that are ready to be sent back to PS for analysis.

- *The AXI DMA* block: this is connected to the *NN* block through an AXI Stream interface for sending/receiving data to/from the *NN* block.
- *The neural Network Architecture* block: this contains the NN model on FPGA. This block is implemented in two approaches: namely, high-level synthesis (HLS) and Verilog implementation.

HLS Approach

The HLS approach builds the NN block from a C implementation. Figure 3.8 describes the NN block's architecture on FPGA. The trained parameters are loaded into BlockRAMs memory, and the input for neurons is extracted from the input stream by the *Pre-processor* module. The hidden and output layers are configured using a pipeline strategy and resource-sharing techniques in which there are only eight neurons in each layer. Additionally, DSP48 blocks are used to implement both MUL and MAC operands. Finally, the *Post-processor* module produces the output stream, which connects to the *AXI DMA* block.

The NN block on FPGA is implemented by Vivado HLS 2018.2. Parameter quan-

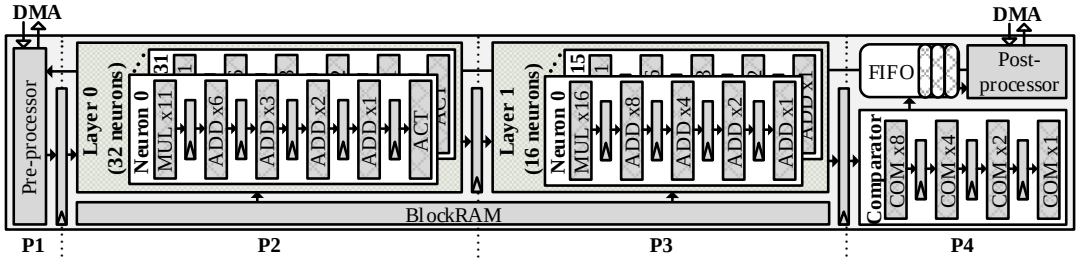


Figure 3.9: Pipeline implementation of the *neural Network Architecture* block on an FPGA-based architecture includes four phases: Input buffering (P1), Layer_0 calculation (P2), Layer_1 calculation (P3), and Output buffering (P4).

tisation and HLS Pragmas techniques are taken into account for implementation, in particular:

- *Parameter Quantisation*: this transforms each trained parameter of an NN model to integer values. For instance, for a layer with weight values in the range of $[-w, w)$, a quantised value, $w_{quantised}$, from w_i, is calculated by

$$w_{quantised} = \lfloor 2^n \frac{w_i}{w} \rfloor \tag{3.5}$$

  where n is the number of bits for storing a weight value in the FPGA. This number is set to 32 in the case of HLS implementation.

- *HLS Pragmas* are applied for optimising the implementation in C++ into the RTL. For instance, *HLS Dataflow* is used for pipelining calculations between layers, *HLS Pipeline* and *HLS Unroll* are used for parallel computation of neurons in each layer.

Verilog Implementation Approach

This approach uses a hardware description language to implement the *Neural Network Architecture* block. Figure 3.9 shows the prototype implementation on an FPGA-based architecture. The data flow is similar to the *HLS implementation approach*; however, the number of neurons in the hidden and output layers is maximised for the designed neural network architecture. In this approach, the pre-trained model on GPU (11 inputs, 32 hidden neurons, 16 output neurons) is selected for implementation.

By using Verilog, the processing performance is optimised manually with a pipeline strategy, in which buffers are placed between each calculation step. The pipelined NN architecture can be described as having four phases:

- *Input buffering* (P1): contains *the Pre-processor* block to receive the eleven

input features from DMA through the AXI4-Stream interface.

- *Layer_0 calculation* (P2): has 32 neurons running simultaneously. These neurons have the same architecture, which has six calculation stages: multiplication, four additions, and an activation. Figure 3.10 illustrates a single neural architecture on FPGA. From the P1 phase, the 11-input features are fed into the corresponding eleven 2-input multipliers (*MULx 11* block). Next, these results are accumulated by the next four addition stages, represented as *ADD x6*, *ADD x3*, *ADD x2*, and *ADD x1* blocks. The final summed result is passed to the *ACT* block, which is implemented as a hardware-efficient ReLU (Rectified Linear Unit) activation function.

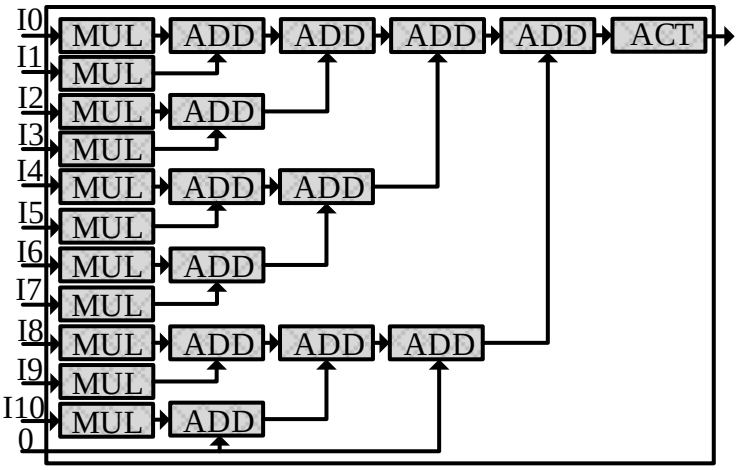


Figure 3.10: The *MULx11* block calculation architecture.

- *Layer_1 calculation* (P3): receives buffered results from the P2 phase; then, these data are distributed to sixteen neurons in this block. Each neuron shares the same calculation architecture with Layer_0 neurons; except that each neuron in Layer_1 has five calculation stages: a multiplication (*MUL x16* block) and four additions (*ADD x8*, *ADD x4*, *ADD x2*, and *ADD x1* blocks). The final results are buffered and will be read in the next clock cycle.

- *Output buffering* (P4): includes the *Comparator*, the *FIFO buffer*, and the *Post-processor* blocks. The *Comparator* block returns the maximum index from sixteen inputs, corresponding to sixteen output neurons in the P3 phase. These indices are stored in a FIFO (First in, first out) buffer. The *Post-processor* block reads data from the buffer and sends it to DMA; this block also sends a signal to the *Pre-processor* block to indicate if the buffer can store all the results.

3.3.5 Results

In this section, the prototype HH-NIDS implementation is evaluated. First, the hardware resource usage on two different platforms: microcontroller (MAX78000EVKIT) and SoC FPGA (PYNQ-Z2), is discussed. Secondly, the accuracy results are discussed and compared to relevant works. Finally, the hardware performance is measured.

3.3.5.1 Implemented Result

On the MAX78000 microcontroller, the NN models have two parameter layers with a total of 1360 parameters, which corresponds to 1360 bytes. With respect to the total hardware resources on the board, 1360 bytes out of 442,368 bytes (0.3%) weight memory were used. Six bytes out of 2048 bytes of total bias memory were utilised (the bias was not used). The records were sent, one after another, repeatedly. Utilising the power monitor built into the MAX78000EVKIT, the power drawn was between 17 mW (loading weights etc.) and 18mW (when the NN model is inferring).

As introduced in Section 3.3.4, the NN models on FPGA are implemented using two different approaches: namely, HLS and Verilog. These implementations on Vivado version 2018.2 were deployed on a PYNQ-Z2 board, which includes Dual ARM Cortex-A9 MPCore and a Xilinx Zynq xc7z020-1clg400c SoC (containing a total of 53,200 LUTs, 106,400 FFs, 140 BRAMs, and 220 DSP slices).

Table 3.3 shows the hardware resource usage, timing and power consumption estimation of the implemented NN for the IoT-23 dataset. In terms of FPGA resource usage, the HLS implementation approach consumed 24.60% LUTs, 15.25% FFs; while the Verilog implementation approach used nearly double these resources, which are 41.97% LUTs and 26.60% FFs on the PYNQ-Z2 board. Although the Verilog implementation approach used fewer BRAMs (14.29% compared to 30.36%), this approach required 220 DSP slices, which is 100% of the on-chip DSP slices on the PYNQ-Z2 board. The maximum frequency is approximately 101MHz in both implemented strategies. The power synthesised in the Verilog implementation is 0.3W less than the HLS implementation (1.53 W compared to 1.83 W).

Similarly, Table 3.4 represents the hardware resource usage, timing and power consumption estimation of the implemented NN models on the UNSW-NB15 dataset. The used LUTs and FFs are 22.15% and 15.26% in the HLS imple-

Table 3.3: The IoT-23 dataset implemented results on PYNQ-Z2 SoC FPGA

Hardware Resources Usage				
Approach	**Resources**	**Utilisation**	**Available**	**Utilisation (%)**
HLS	LUT	13,089	53,200	24.60
	LUTRAM	242	17,400	1.39
	FF	16,224	106,400	15.25
	BRAM	42.5	140	30.36
	DSP	152	220	69.09
Verilog	LUT	22,329	53,200	41.97
	LUTRAM	1420	17,400	8.16
	FF	28,304	106,400	26.60
	BRAM	20	140	14.29
	DSP	220	220	100
Timing and Power				
	F_max	**Static**	**Dynamic**	**Total**
HLS	102.1 MHZ	0.15 W	1.68 W	1.83 W
Verilog	101.2 MHZ	0.14 W	1.39 W	1.53 W

mentation approach; these numbers are 52.64% and 31.93% in the Verilog implementation approach, respectively. The BRAMs and DSP resources used are approximately equal to those of the IoT-23 dataset implementations. The maximum frequency is approximately 102.4 MHz in both approaches, while the power consumption in the Verilog implementation is 0.23 W less than in the HLS implementation approach.

3.3.5.2 Accuracy

The training, validation and inference accuracy are discussed in this section. There are two training schemes corresponding to the targeted hardware: MAX78000EVKIT and PYNQ-Z2. The training processes were run in each scenario with three different learning rates (LR): 0.0005, 0.001, and 0.01 on the UNSW-NB15 and IoT-23 datasets.

Figure 3.11 depicts the training and validation accuracy in the first scheme, which was aimed at the MAX78000EVKIT. In detail, the line chart in Figure 3.11a shows trained results on the UNSW-NB15 dataset. The highest LR (0.01) returned the best score at 97.96% accuracy (epoch 8). However, this

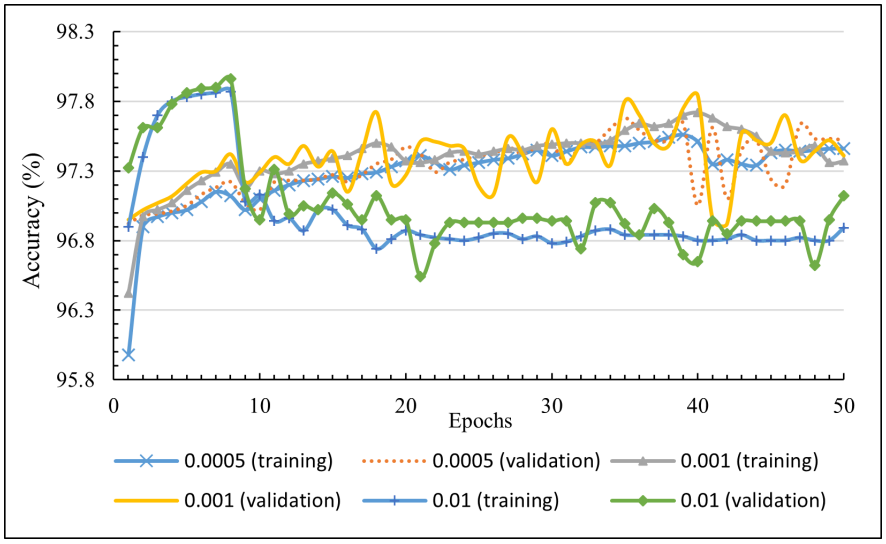
Table 3.4: The UNSW-NB15 dataset implemented results on PYNQ-Z2 SoC FPGA

Hardware Resources Usage				
Approach	**Resources**	**Utilisation**	**Available**	**Utilisation (%)**
HLS	LUT	11,783	53,200	22.15
	LUTRAM	242	17,400	1.39
	FF	16,236	106,400	15.26
	BRAM	44	140	31.43
	DSP	152	220	69.09
Verilog	LUT	28,004	53,200	52.64
	LUTRAM	1412	17,400	8.11
	FF	33,974	106,400	31.93
	BRAM	20	140	14.29
	DSP	219	220	99.55
Timing and Power				
	F_max	**Static**	**Dynamic**	**Total**
HLS	102.5 MHZ	0.15 W	1.64 W	1.79 W
Verilog	102.4 MHZ	0.14 W	1.42 W	1.56 W

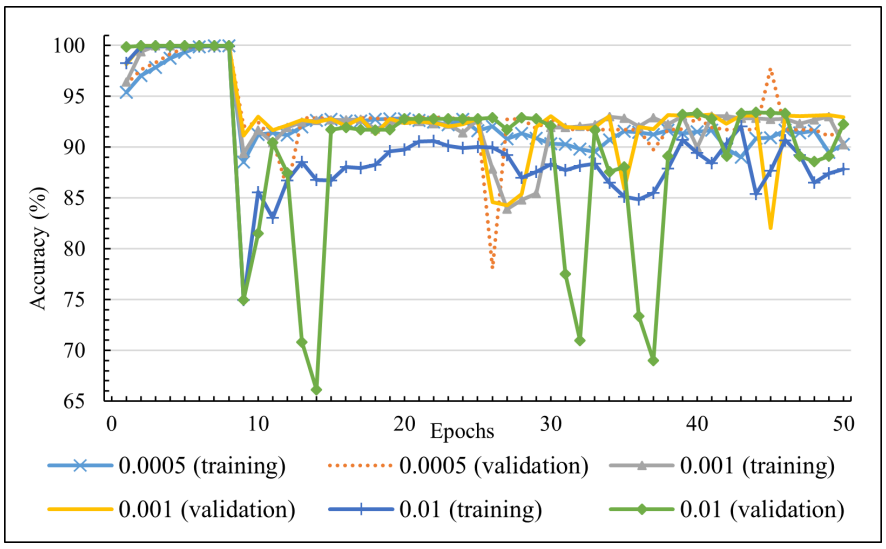
number dropped significantly, to approximately 96.82%, in the rest of the training process, after the quantisation-aware training was activated. The training processes with lower LR reported better results, which drew increasing trends in validation accuracy, from 97.12% to the peak at 97.84% from epoch 9 to epoch 40 (with LR at 0.001).

Figure 3.11b illustrates the training and validation accuracy on the IoT-23 dataset. The accuracy can reach up to 99.98% before dropping dramatically to 89.38% (0.001 LR) and 74.94% (0.01 LR) due to the activation of quantisation-aware training. The 0.0005 and 0.001 LR training processes showed a slight improvement over time, with training accuracy around 92.85%. The lower LR values (0.0005 and 0.001) produced better models in the quantisation-aware training processes. The accuracy fluctuated if LR was 0.01 on both datasets.

In the second scheme, the training processes were aimed at the PYNQ-Z2 board. Figure 3.12 represents two charts showing accuracy results on the two datasets. The line chart in Figure 3.12a shows trained results on the UNSW-NB15 dataset. The 0.01 LR reported 96.84% accuracy in training and validation at epoch one, before increasing to 97.71% in the subsequent seven epochs; this number fluctuated slightly in the rest of the training process. The 0.001 LR also reported



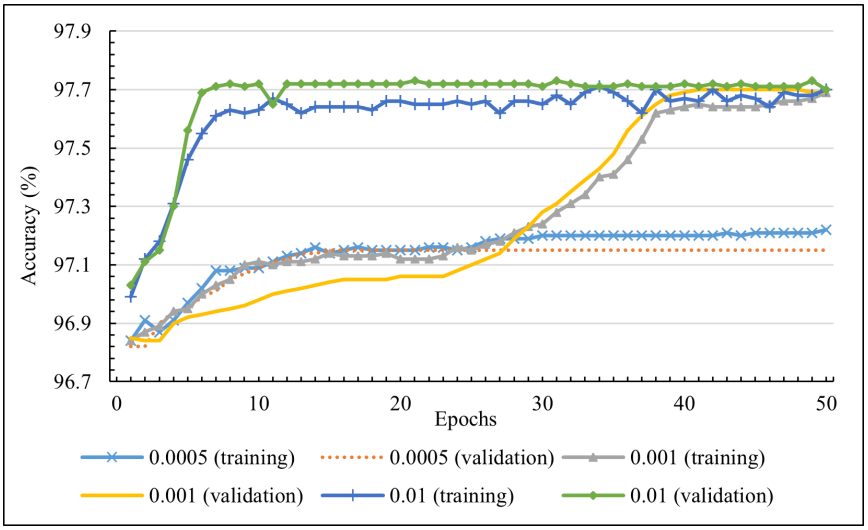
(**a**) Results on UNSW-NB15 dataset.



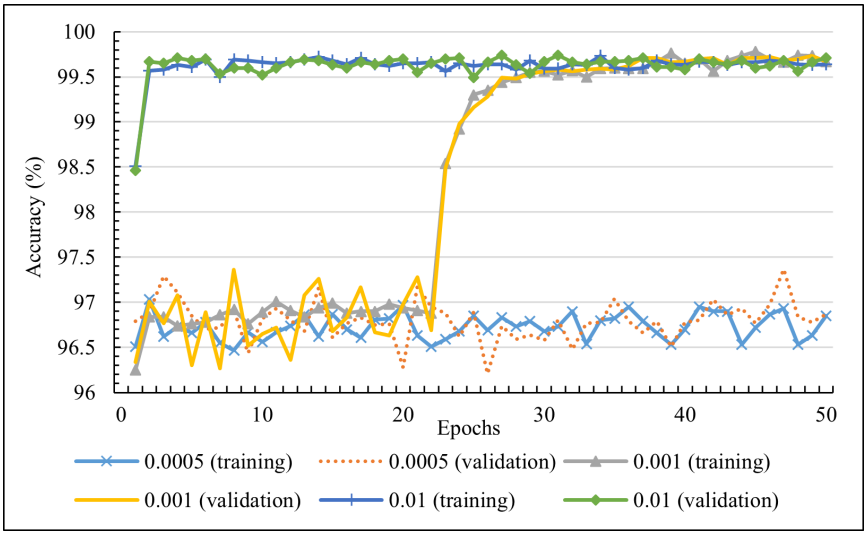
(**b**) Results on IoT-23 (2) dataset.

Figure 3.11: The trained results in fifty epochs with three different learning rates: 0.0005, 0.001, and 0.01 for the MAX78000 microcontroller.

the same results, with the best validation accuracy being 97.70% from epoch 40, while the smallest LR (0.0005) can only reach 97.15% accuracy at epoch 50. The second line chart, which are trained results on the IoT-23 dataset, shows similar training patterns to the first line chart. In detail, the higher LR (0.001 and 0.01) reported the highest validation accuracy at around 99.71%, while the 0.0005 LR training process achieved the best accuracy of 96.37% at epoch 47. The training and validation accuracies when using the medium LR (0.001) have been improved significantly from epoch 23 to 30.



(**a**) Results on UNSW-NB15 dataset.



(**b**) Results on IoT-23 dataset.

Figure 3.12: The trained results in fifty epochs with three different learning rates: 0.0005, 0.001, and 0.01 for the PYNQ-Z2 SoC FPGA.

The four best models, in terms of validation accuracy, were collected from the trained results above. Evaluation metrics were calculated, including accuracy, precision, recall and F1-score on test data. Table 3.5 shows inference results on the MAX78000EVKIT on two models, which are denoted as (1) and (2).

Table 3.5: The inference results (in %) for the two trained models on the MAX78000EVKIT.

Dataset	Accuracy	Precision	Recall	F1-score
UNSW-NB15 (1)	98.57	90.11	97.09	93.47
IoT-23 (2)	92.69	95.04	95.22	95.13

Model (1) on the UNSW-NB15 dataset achieves 98.57% accuracy and a 93.47% F1-score. The precision is 6.98% lower than the recall value (90.11% compared to 97.09%), which means that the model is more sensitive to attack records than normal records. In addition, model (2) on the IoT-23 dataset reports 92.69% accuracy and, 95.13% F1-score.

Accordingly, Table 3.6 represents inference results on the PYNQ-Z2 board from two models, which are denoted as (3) and (4).

Table 3.6: The inference results (in %) for the two trained models on the PYNQ-Z2 board.

Dataset	Accuracy	Precision	Recall	F1-score
UNSW-NB15 (3)	98.43	87.95	98.01	92.71
IoT-23 (4)	99.66	99.97	99.65	99.81

Model (3) reproduces similar results to model (1) on the UNSW-NB15 dataset, with 98.43% accuracy and 92.71% F1-score. The highest accuracy and F1-score (99.66% and 99.81%) are achieved by the model (4) on the IoT-23 dataset.

The trained NN models on the IoT-23 dataset are compared to related works in Table 3.7.

Table 3.7: Result comparison to related works in the IoT-23 dataset

System	Model	Accuracy (%)	F1-score (%)
(Dutta et al. 2020a)	DNN	99.70	87.00
(Dutta et al. 2020b)	LSTM	99.99	N/A
(Hussain et al. 2020)	RF	100.00	100.00
(Hegde et al. 2020)	NN	99.80	99.85
(Thamaraiselvi and Mary 2020)	RF	99.50	99.00
HH-NIDS	NN	99.66	99.81

The authors in (Dutta et al. 2020a) have achieved 99.70% accuracy by using DNN. They have improved this result to 99.99% by applying LSTM in (Dutta et al. 2020b); however, F1-score is not mentioned in this research. Authors in (Hussain et al. 2020; Thamaraiselvi and Mary 2020) have reported RF as the best algorithm for detecting attacks with 100% and 99.50% accuracy, respectively. Authors in (Hegde et al. 2020) have achieved an accuracy of only 0.14% greater than the implemented model. However, their experiments have not applied to the full IoT-23 dataset (Hussain et al. 2020; Hegde et al. 2020).

3.3.5.3 Performance

Inference time is one of the most critical metrics for evaluating the HH-NIDS framework. In this section, performance measurements will be performed on the four hardware platforms, including Intel® Core™ i7-9750H 2.6GHz CPU, NVIDIA GeForce GTX 1650 GPU, MAX78000EVKIT and PYNQ-Z2 SoC FPGA boards.

Inference Time

The first test measured pre-process time (allocates input buffers), inference time, and post-process time (for receiving results) of each implementation when only one input is sent. Figure 3.13 illustrates the five implementations' results. The bar chart shows that the inference time on the MAX78000EVKIT is only 15 μs, which is 11.3 and 21.3 times faster than CPU and GPU approaches, respectively. This number outperformed all other platforms in inference time; however, the pre-process and post-process times are not measured due to unsupported library issues. Besides, CPU and GPU implementations report the lowest pre-process and post-process time; these numbers are approximately 0.02 ms, thanks to high-frequency host processors. The inference time on CPU is 0.17 ms, while these times are 0.32 ms and 0.43 ms on GPU and FPGA, respectively. The GPU used in this experiment exhibited slower processing performance compared to the CPU due to the input size being limited to a single record. This outcome arises from the fact that the strength of GPUs lies in their ability to process multiple inputs in parallel.. The FPGA implementations (HLS and Verilog) have higher inference times (between 0.45 ms and 0.46 ms) because of the overhead in data transfer time from the PS to the PL.

FPGA Performance

The PL configuration time is 340ms on average, including the time for loading

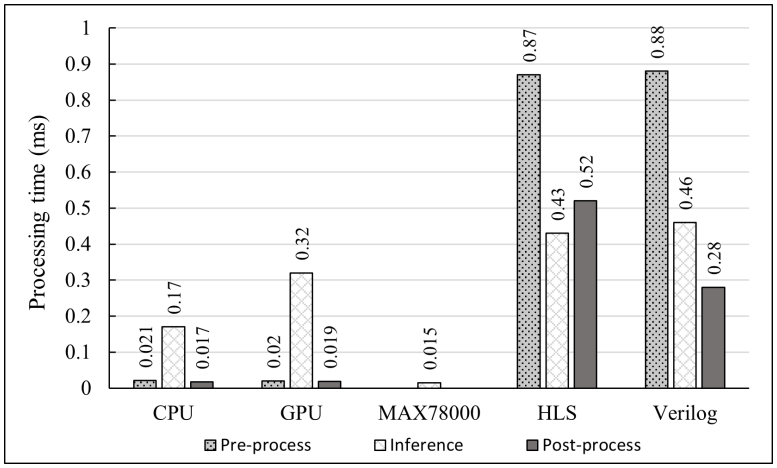


Figure 3.13: The pre-process, inference, and post-process time from different implementations (except the MAX78000EVKIT, see text).

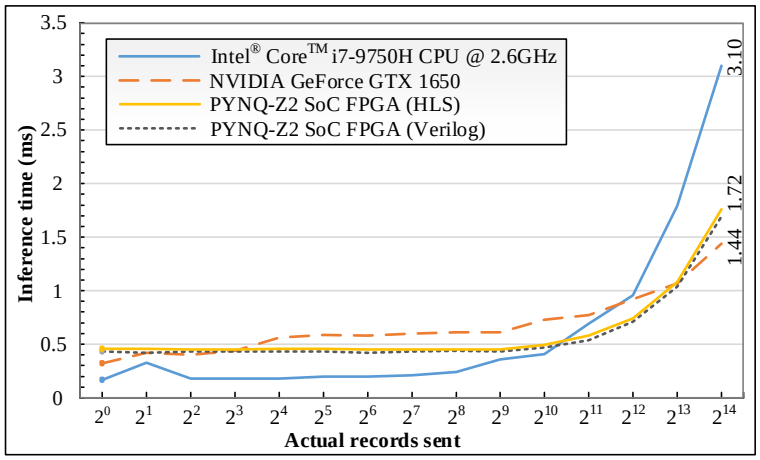


Figure 3.14: The CPU, GPU, and FPGA inference times for different input buffer sizes.

NN parameters into BlockRAM. The line graph in Figure 3.14 shows the differences in inference times between CPU, GPU and FPGA. The horizontal and vertical axes present the number of records per test (input buffer size) and the processing time, respectively. By doubling the number of records sent each time, the CPU and GPU inference times increase gradually to approximately 0.92 ms, for both platforms, at the buffer size of 2^{12} (4096 records). The CPU inference time rises exponentially to 3.01 ms, while the GPU starts to perform better, with only 1.44 ms, at the end of the test. HLS and Verilog implementation approaches on the PYNQ-Z2 SoC FPGA return a similar inference time, which starts at approximately 0.44 ms in the beginning and then increases slightly to 0.48 ms at the buffer size of 1024. This time is approximately 1.72 ms when the buffer size is 16,384 (2^{14}), which is equal to 9,525,581 records per second.

Next, a timer is placed in the inference code after DMA has sent data from the PS to the PL to measure the data transfer time.

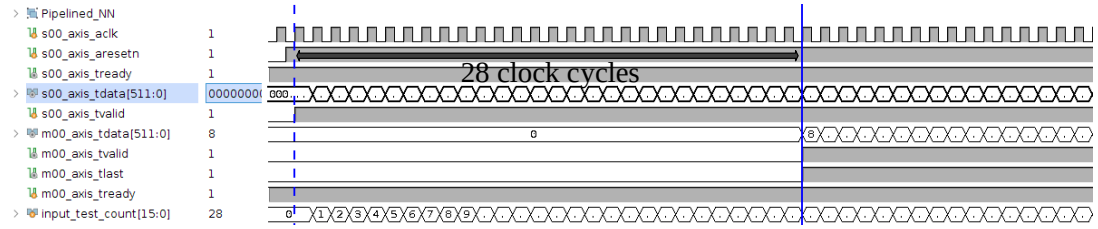


Figure 3.15: The wareform simulation results from the NN on FPGA using a Verilog approach.

The timer reported that it takes 0.067 ms, on average, for an input on the PYNQ-Z2 Soc FPGA to be inferred and transferred back to the PS. In addition, the data transfer time from the PS to the PL is close to 0.42 ms. Therefore, the NN block on PYNQ-Z2 SoC, with a Xilinx xc7z020-1clg400c device, is 2.5 and 4.8 times (0.067 ms compared to 0.17 ms and 0.32 ms) faster than Intel® Core™ i7-9750H 2.6GHz CPU and NVIDIA GeForce GTX 1650 GPU in inference time, respectively. The processing speed of the pipelined design on FPGA can be calculated through the waveform simulation result in Figure 3.15.

The waveform displays the primary input and output AXI4-STREAM interfaces in the NN block. The testbench sends a new input at each clock cycle to demonstrate a network input flow. Each record takes 28 clock cycles to be processed (between the s00_axis_tvalid and m00_axis_tvalid signals). In this pipeline design, the input flow can go through the system continuously without waiting for other processing inputs in the system to be finished; thus, the system overhead for processing one flow is 28 clock cycles, which is equal to 0.28 μs on the PYNQ-Z2 (operating at 100 MHz). This means that the pipelined NN block on the FPGA accelerator at 100 MHz is 53.5 times faster than the max78000EVKIT in simulation. Some factors affected the inference time, in practice, and will be discussed in Section 3.3.6.

Finally, the average inference times after ten trials are shown in Table 3.8. To ensure a fair comparison with the work in (Ioannou and Fahmy 2019), the number of records for sending in DMA buffer size was made equal to the number in these works (22,544 records). The authors in (Ioannou and Fahmy 2019; Idhammad, Afdel, and Belouch 2017) used the same NSL-KDD dataset for experimenting with their NN models. The FPGA implementation is 4.15 times faster than the same platform in (Ioannou and Fahmy 2019) and 110.1

Table 3.8: Comparison in inference time on different platforms and between related works

System	Platform	Time (ms)
(Idhammad, Afdel, and Belouch 2017)	CPU i3 2.4GHz	240.14
(Ioannou and Fahmy 2019)	ARM A9 667 MHz	1458.10
	FPGA accelerator 76 MHz	9.02
HH-NIDS	**FPGA accelerator 100 MHz (HLS)**	**2.18**
	FPGA accelerator 100 MHz (Verilog)	**2.24**

times faster than the CPU implementation in (Idhammad, Afdel, and Belouch 2017).

3.3.6 Discussion

The HH-NIDS framework has trained lightweight NN models on the UNSW-NB15 and IoT-23 datasets using different schemes aimed at two hardware acceleration platforms. Although the eleven inputs used in this prototype are static features, the trained models have achieved high accuracy compared to the related works. To improve the results (higher accuracy and lower the false alarm rate), the Data Pre-processing block in the HH-NIDS framework needs to be upgraded to extract a more comprehensive input feature set.

In Section 3.3.5.3, CPU and GPU implementations have an advantage in pre-process and post-process times thanks to high operating frequencies and rich memories for buffering input data; the data transfer buses in these platforms are also optimised for a broader range of applications. The GPU also has extra overhead for loading data into its on-chip memory, which explains why it starts to perform better with a greater input buffer size setup.

The MAX78000 microcontroller with NN acceleration processors performs well, requiring only 15 μs to infer an input. The power consumption is ultra-low at 18 mW, while it draws from 15 W to 17 W on the NVIDIA GeForce GTX 1650 GPU to infer inputs. Besides, the NN models on FPGA have shown high overhead in data transferring between the PS and the PL; the actual inference time was explained in the waveform simulation results in Section 3.3.5.3.

While the HLS implementation is more flexible in changing the NN architecture, the Verilog implementation with pipelining is optimised for the best processing performance. Even though the pipelined design on the PL has no bottle-neck points that possibly operate at the on-chip frequency with a fixed 28 clock cycles overhead, the pre-process and post-process blocks on the PYNQ-Z2 SoC FPGA

are run on the PS (ARM-based) with limited speed.

3.4 Summary

This chapter studied the background knowledge of IoT networks, focusing on two main aspects: IoT architecture and protocols. Besides, attacks on IoT networks are discussed to provide an overview of potential vulnerabilities and the associated security countermeasures.

Then, different hardware acceleration platforms are studied for IoT security, especially focusing on platforms with rich computational resources, low power consumption, and the ability to run lightweight machine learning models. The specific hardware platforms that are experimented with in this thesis, including the Max78000 AI-assisted microcontroller, the PYNQ-Z2 FPGA SoC, and the ALVEO U280 FPGA card, are discussed.

This chapter presented the HH-NIDS framework for a heterogeneous hardware-based implementation of anomaly detection in IoT networks. The ML models, which are pre-trained in Chapter 2, are used to implement on the accelerated hardware platforms. The framework includes dataset pre-processing, model generation and hardware-based inference on hardware accelerators.

The proposed framework is tested with the UNSW-NB15 and IoT-23 datasets using NN models and achieves the highest accuracy of 99.66%. The inference phase is implemented using different approaches on the microcontroller and SoC FPGA. Firstly, the inference phase is implemented on the MAX780000EVKIT, which is 11.3 and 21.3 times faster than Intel® Core™ i7-9750H 2.6GHz CPU and NVIDIA GeForce GTX 1650 GPU, respectively; the power drawn was between 17mW and 18mW when the NN model is inferring. In addition, the inference phase on SoC FPGA is implemented using HLS and Verilog approaches, achieving the same practical processing speed. However, the pipelined design on the PYNQ-Z2 board with the Xilinx Zynq xc7z020-1clg400c device has been optimised to run at the on-chip frequency with a fixed 28 clock cycles overhead. The simulation results have reported a speedup of 53.5 times compared to the MAX78000EVKIT.

By analysing the experimental results in this chapter, it is clear that accelerating lightweight ML models on hardware is a promising approach in IDSs. There are two developmental directions for enhancing the HHNIDS framework in the

following chapters:

Firstly, the network-specialised hardware (ALVEO U280 FPGA card) is considered for studying the next version of the HH-NIDS framework. This card allows packets to be received and processed directly on the FPGA fabric, erasing the overhead in data transfer between platforms encountered in the experiments using the PYNQ-Z2 FPGA SoC.

Secondly, the next chapter will explore CNN models for network attack detection. The ANN model has limitations as the input data consists only of header fields, and extracting additional data can cause extra overhead. Furthermore, the packet payload remains unused. By utilising CNN models, the network packet will be converted into images; this process eliminates the logic in the data extraction process, thus reducing the complexity of data pre-processing. Additionally, the packet payload will be directly contributed as input to the CNN model equally.

References

Ahli, Alia et al. (2023). "Binary and Multi-Class Classification on the IoT-23 Dataset". In: *2023 Advances in Science and Engineering Technology International Conferences (ASET)*. IEEE, pp. 1–7.

Al, Alaa Abdul Al Muhsen Hussain et al. (2024). "IoT Maleware Detection Based On Anomaly Traffic Identification Using CNN-LSTM". In: *2024 16th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE, pp. 1–6.

Alajlan, Razan, Norah Alhumam, and Mounir Frikha (2023). "Cybersecurity for blockchain-based IoT systems: a review". In: *Applied Sciences* 13.13, p. 7432.

Alani, Mohammed M and Ali Miri (2022). "Towards an Explainable Universal Feature Set for IoT Intrusion Detection". In: *Sensors* 22.15, p. 5690.

Ali, Omer et al. (2022). "A comprehensive review of internet of things: Technology stack, middlewares, and fog/edge computing interface". In: *Sensors* 22.3, p. 995.

AMD (2021). *AMD ALVEO U280: Adaptable Accelerator Cards for Data Center Workloads*. Accessed: Dec 16, 2024.

Antonopoulos, Christos P and Nikolaos S Voros (2017). "A data compression hardware accelerator enabling long-term biosignal monitoring based on ultra-low power IoT platforms". In: *Electronics* 6.3, p. 54.

Chéour, Rym, Sabrine Khriji, Olfa Kanoun, et al. (2020). "Microcontrollers for IoT: optimizations, computing paradigms, and future directions". In: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE, pp. 1–7.

Cisco (2014). *The Internet of Things: Capturing the Accelerated Opportunity*. https://blogs.cisco.com/digital/the-internet-of-things-capturing-the-accelerated-opportunity. Accessed: Dec 16, 2024.

d'Orazio, Laurent and Julien Lallet (2018). "Semantic caching framework: An FPGA-based application for IoT security monitoring". In: *Open Journal of Internet Of Things (OJIoT)* 4.1, pp. 150–157.

Douiba, Maryam et al. (2022). "An improved anomaly detection model for IoT security using decision tree and gradient boosting". In: *The Journal of Supercomputing*, pp. 1–20.

Dutta, Vibekananda et al. (2020a). "A deep learning ensemble for network anomaly and cyber-attack detection". In: *Sensors* 20.16, p. 4583.

– (2020b). "Detection of Cyberattacks Traces in IoT Data". In: *Journal of Universal Computer Science* 26.11, pp. 1422–1434.

Gao, C. et al. (2019). "Real-Time Speech Recognition for IoT Purpose using a Delta Recurrent Neural Network Accelerator". In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5. DOI: 10.1109/ISCAS.2019.8702290.

Garcia, S, A Parmisano, and MJ Erquiaga (2020b). "IoT-23: A labeled dataset with malicious and benign IoT network traffic". In: *Stratosphere Lab., Praha, Czech Republic, Tech. Rep*.

Google (2024). "Ask a Techspert: What's the difference between a CPU, GPU and TPU?" In: *Google Blog*. URL: https://blog.google/technology/ai/difference-cpu-gpu-tpu-trillium/.

Gupta, Brij B and Megha Quamara (2020). "An overview of Internet of Things (IoT): Architectural aspects, challenges, and protocols". In: *Concurrency and Computation: Practice and Experience* 32.21, e4946.

Hegde, Mandira et al. (2020). "Identification of Botnet Activity in IoT Network Traffic Using Machine Learning". In: *2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*. IEEE, pp. 21–27.

Hossin, Mohammad and Md Nasir Sulaiman (2015). "A review on evaluation metrics for data classification evaluations". In: *International journal of data mining & knowledge management process* 5.2, p. 1.

Hussain, Faisal et al. (2020). "Towards a Universal Features Set for IoT Botnet Attacks Detection". In: *arXiv preprint arXiv:2012.00463*.

Idhammad, Mohamed, Karim Afdel, and Mustapha Belouch (2017). "Dos detection method based on artificial neural networks". In: *International Journal of Advanced Computer Science and Applications* 8.4, pp. 465–471.

Impulse, Edge (2025). *The Convergence of Edge AI and Cloud: Making the Right Choice for Your AI Strategy*. URL: https://www.edgeimpulse.com/blog/edge-ai-vs-cloud-computing-making-the-right-choice-for-your-ai-strategy/.

Ioannou, Lenos and Suhaib A Fahmy (2019). "Network intrusion detection using neural networks on FPGA SoCs". In: *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, pp. 232–238.

Jing, Qi et al. (2014). "Security of the Internet of Things: perspectives and challenges". In: *Wireless networks* 20, pp. 2481–2501.

Kalantar, Amin, Zachary Zimmerman, and Philip Brisk (2021). "FA-LAMP: fpga-accelerated learned approximate matrix profile for time series similarity prediction". In: *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, pp. 40–49.

Khan, Minhaj Ahmad and Khaled Salah (2018). "IoT security: Review, blockchain solutions, and open challenges". In: *Future generation computer systems* 82, pp. 395–411.

Korba, Abdelaziz Amara, Aleddine Diaf, and Yacine Ghamri-Doudane (2024). "AI-Driven Fast and Early Detection of IoT Botnet Threats: A Comprehensive Network Traffic Analysis Approach". In: *2024 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, pp. 1779–1784.

Krishna, Ritika Raj et al. (2021). "State-of-the-art review on IoT threats and attacks: Taxonomy, challenges and solutions". In: *Sustainability* 13.16, p. 9463.

Kumar, Sachin, Prayag Tiwari, and Mikhail Zymbler (2019). "Internet of Things is a revolutionary approach for future technology enhancement: a review". In: *Journal of Big data* 6.1, pp. 1–21.

Kumar, Sudeendra et al. (2017). "Security enhancements to system on chip devices for IoT perception layer". In: *2017 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*. IEEE, pp. 151–156.

Lallet, Julien, Emmanuel Casseau, Laurent d'Orazio, et al. (2020). "Mascara (ModulAr semantic caching framework) towards FPGA acceleration for IoT security monitoring". In: *International Workshop on Very Large Internet of Things (VLIoT 2020)*.

Liang, Qianlin, Prashant Shenoy, and David Irwin (2020). "AI on the Edge: Rethinking AI-based IoT Applications Using Specialized Edge Architectures". In: *arXiv preprint arXiv:2003.12488*.

Maitra, S. et al. (2019). "Performance Evaluation of IoT Encryption Algorithms: Memory, Timing, and Energy". In: *2019 IEEE Sensors Applications Symposium (SAS)*, pp. 1–6. DOI: 10.1109/SAS.2019.8706017.

Mark Russinovich (2017). *Inside the Microsoft FPGA-based configurable cloud*. https://www.microsoft.com/en-us/research/video/inside-microsoft-fpga-based-configurable-cloud/ visited on Dec 16, 2024.

Maxim Integrated (2020a). *MAX78000 - Artificial Intelligence Microcontroller with Ultra-Low-Power Convolutional Neural Network Accelerator*. https://www.maximintegrated.com/en/products/microcontrollers/MAX78000.html visited on Dec 16, 2024.

– (2020b). *MAX78000EVKIT - Evaluation Kit for the MAX78000*. https://www.maximintegrated.com/en/products/microcontrollers/MAX78000EVKIT.html visited on Dec 16, 2024.

Microsoft (2021). *Deploy ML models to field-programmable gate arrays (FPGAs) with Azure Machine Learning*. https://docs.microsoft.com/en-us/azure/

machine-learning/how-to-deploy-fpga-web-service visited on Dec 16, 2024.

Moustafa, Nour and Jill Slay (2015). "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 military communications and information systems conference (MilCIS)*. IEEE, pp. 1–6.

Nazir, Ahsan et al. (2024). "A deep learning-based novel hybrid CNN-LSTM architecture for efficient detection of threats in the IoT ecosystem". In: *Ain Shams Engineering Journal* 15.7, p. 102777.

Nobakht, Mehrnoosh, Reza Javidan, and Alireza Pourebrahimi (2022). "DEMD-IoT: a deep ensemble model for IoT malware detection using CNNs and network traffic". In: *Evolving Systems*, pp. 1–17.

Othman, Trifa Sherko and Saman Mirza Abdullah (2023). "Machine Learning Techniques Evaluation with SMOTE on IoT-23 Dataset". In: *2023 9th International Engineering Conference on Sustainable Technology and Development (IEC)*. IEEE, pp. 7–13.

Santos, Marcela G dos et al. (2020). "Internet of Things architectures: A comparative study". In: *arXiv preprint arXiv:2004.12936*.

Stoian, Nicolas-Alin (2020). "Machine Learning for anomaly detection in IoT networks: Malware analysis on the IoT-23 data set". B.S. thesis. University of Twente.

Storcheus, Dmitry, Afshin Rostamizadeh, and Sanjiv Kumar (2015). "A survey of modern questions and challenges in feature extraction". In: *Feature Extraction: Modern Questions and Challenges*. PMLR, pp. 1–18.

Tchagna Kouanou, Aurelle et al. (2022). "Securing data in an internet of things network using blockchain technology: smart home case". In: *SN Computer Science* 3.2, p. 167.

Thamaraiselvi, D and S Mary (2020). "Attack and anomaly detection in iot networks using machine learning". In: *Int. J. Comput. Sci. Mob. Comput* 9.10, pp. 95–103.

Tudosa, Ioan et al. (2019). "Hardware security in IoT era: The role of measurements and instrumentation". In: *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4. 0&IoT)*. IEEE, pp. 285–290.

Ullah, Imtiaz and Qusay H Mahmoud (2022). "Design and Development of RNN Anomaly Detection Model for IoT Networks". In: *IEEE Access* 10, pp. 62722–62750.

Wielgosz, Maciej and Michał Karwatowski (2019). "Mapping neural networks to FPGA-based IoT devices for ultra-low latency processing". In: *Sensors* 19.13, p. 2981.

Xilinx (2020). *XUP PYNQ-Z2*. `https://www.xilinx.com/support/university/xup-boards/XUPPYNQ-Z2.html` visited on Dec 16, 2024.

Chapter 4

Deep Packet Inspection: A Convolutional Neural Networks Approach

The previous chapter proposed a security framework for IoT networks using lightweight ANN models on accelerated hardware platforms. While using ANN models on a set of selected features extracted from packet headers has the advantages of low computational cost and complexity, this approach ignores other headers and payloads in a packet. Furthermore, a network packet is typically diverse in structure due to the existence of multiple protocols with different versions. This chapter studies SVM and CNN in IDSs, focusing on packet processing methodologies, efficiency, and flexibility for hardware implementations.

The IoTs has contributed a significant volume of data exchange to the global network in the past decade. Due to the high demand for smart devices, the number of IoT devices is increasing significantly every year, resulting in the need to properly manage their data. Besides, with the ever-increasing dependence on digital infrastructure across industries, organisations must develop robust cybersecurity measures to mitigate network attacks and protect users' information.

Network administrators have made significant progress in integrating security functions into the system to enhance device security against attacks. These functions can be built based on signature-based or anomaly-based approaches, as well as a hybrid approach that combines both. This chapter continues to explore anomaly-based IDS approaches. In detail, this chapter proposes 2D-

CNN (Two-dimensional Convolutional Neural Network) models for deep packet inspection. The proposed approach involves labelling network packets in the IoT-23 dataset from pre-trained ANN models; then, 2D-CNN models are trained to detect attacks at the packet level.

This chapter utilises ANN and CNN on raw network packets to detect network attacks. The chapter's contributions can be summarised as follows:

- The previous work (Ngo, Lightbody, Temko, Pham-Quoc, et al. 2023a) (HH-NIDS framework - Chapter 3.3) with the pre-trained ANN models is used to assign labels to the raw network packets in the IoT-23 dataset.
- The CNN models are trained on the labelled packets. This approach mitigates the computational overhead of the feature extraction process associated with the ANN.
- The experiments are conducted to evaluate the accuracy and performance of the proposed approach.

The rest of the chapter is organised as follows. Section 4.1 provides background knowledge and discusses the background and related work. Section 4.2 presents the proposed approach in detail. The experimental results of applying the proposed approach to the IoT-23 dataset are presented in Section 4.3. Finally, Section 4.4 concludes the chapter and discuss potential directions for the next steps.

4.1 Background and Related Work

4.1.1 Deep Packet Inspection

The previous chapter introduced two network intrusion detection systems based on the algorithms and detection models used: signature-based and anomaly-based IDSs. This chapter studies an overview of Deep Packet Inspection (DPI) history and approaches. While traditional forms of packet filtering look only at packet headers, DPI explores the payload, which can extract more information and has a better understanding of network traffic. DPI plays an important role in both security and traffic management in networking systems.

According to authors in (Lotfollahi et al. 2020), DPI approaches can be fitted into three main categories: port-based filtering, payload inspection and ML

approaches. The three approaches are correspondingly mapped with DPI's evolution over time.

- Port-based filtering uses the port fields (source and destination) in the packet header to identify the associated application to which the packets belong. The extraction process is straightforward since the port fields are not encrypted. Port-based filtering was used to implement access control lists in firewalls (Qi et al. 2009). This stage is considered the early development of DPI.
- Payload inspection approaches look into the data of network packets. The most common approaches compare the extracted payload to a set of stored signatures (Dainotti, Pescape, and Claffy 2012). The payload examination technique is reliable for traffic classification and has been used by companies like Cisco Systems (Snort), Juniper, and Ericsson, just to name a few; however, there are privacy challenges which prevent access to the packet's content. With the rise of secured communication protocols (HTTPs, TLS) in the 2010s, the payload became encrypted, causing challenges for traditional inspection techniques. New techniques require decrypting data or inspecting encrypted payload without decryption (Sherry et al. 2015).
- ML approaches show advantages compared to traditional approaches because they can use mathematical models to inspect packets instantly without querying databases, which is required when using pattern-matching techniques. Furthermore, ML-based DPI can identify traffic patterns and new threats in large, complex networks.

The development of DPI approaches has gone through various stages. They are closely tied to the changes in the internet infrastructure. The section provides an observation on how these DPI systems have evolved, from simply using the pattern-matching technique to applying ML to adapt to new network architectures, protocols, and applications. With the explosion of AI and ML, the advancement of DPI systems has accelerated even further. The research papers on DPI techniques, utilising deep learning on packet payload for traffic classification and attack detection, are discussed below.

In 2015, (Velan et al. 2015) surveyed methods for classifying and analysing encrypted network traffic from 2005 to 2014. They summarised the packet classification process through three main stages, namely *classification input*,

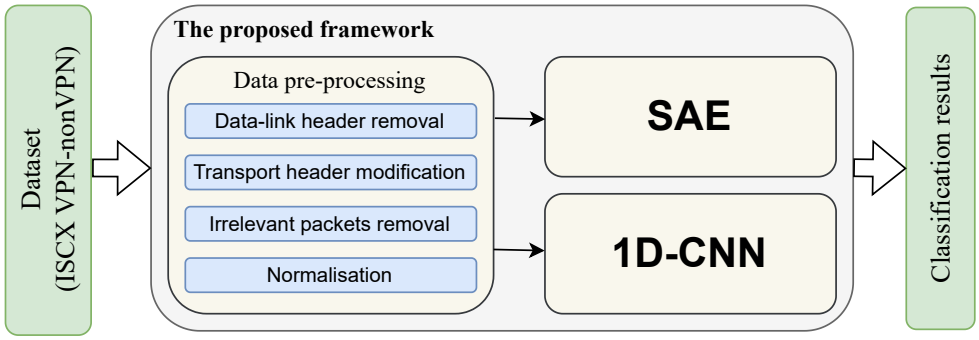


Figure 4.1: The proposed framework in (Lotfollahi et al. 2020) combining SAE and CNN models for traffic classification.

classification technique, and *classification output*. It can be seen from their results that other researchers mainly focused on feature-based methods since they showed efficient and comparable detection rates. The raw packet data is useful for recognising the applications associated with the packet.

The research in (Lotfollahi et al. 2020) is one of the pioneers in using ML models to inspect network packets. The approach eliminates the need to use predefined features extracted by an expert to classify network traffic. They propose a deep learning-based approach, which integrates both feature extraction and classification phases into one system. The proposed framework combines two ML models, namely Stacked Autoencoder (SAE) and CNN, to classify network traffic into major classes. Figure 4.1 explains the proposed framework's methodology.

The proposed framework involves a data pre-processing module to convert raw packets into a suitable format for processing in ML models, SAE and CNN. During the data pre-processing phase, they first eliminate header fields that are irrelevant for packet classification. Next, they standardise the transport layer segment of UDP and TCP packets to a fixed number (20 bytes). Additionally, irrelevant packets, such as those related to three-way handshake protocols and DNS resolution packets, are removed from the dataset. Once the data is balanced, they normalise it again by padding short packets with zero to a fixed size before splitting the data to train SAE and 1D-CNN models.

The ISCX VPN-nonVPN dataset (Cybersecurity 2016) is used to evaluate the implemented system. Their CNN models have achieved recalls of 0.98 and 0.94 in the application identification and traffic categorisation tests, respectively. Importantly, they discussed how CNN models are promising for the network traffic

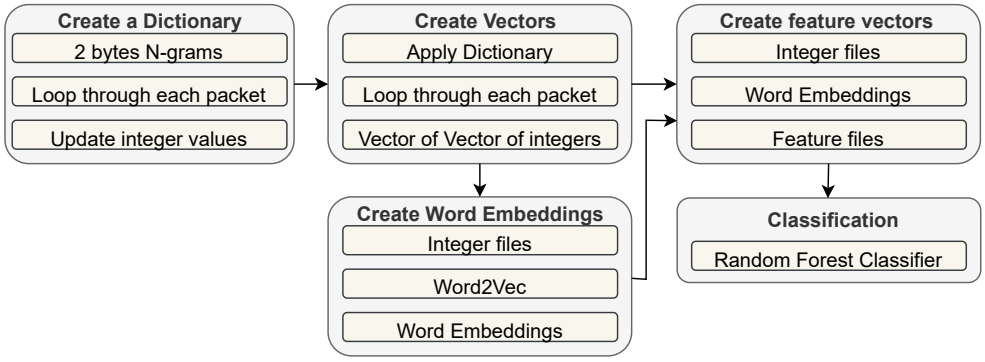


Figure 4.2: The Packet2Vec approach's process flow.

classification task. CNN models can capture spatial dependencies between adjacent bytes so that discriminative patterns can be found.

The work in (Goodman, C. Zimmerman, and Hudson 2020) proposes a Packet2Vec (Packet to Vector) approach for detecting malicious traffic on the 2009 DARPA network dataset (Gharaibeh and Papadopoulos 2014). The core of Packet2Vec is reported to be modified from the Word2Vec (Mikolov 2013), a natural language processing project. The approach creates a set of vectors to represent words so that similar words are positioned close to each other. For example, given three vectors, "*King*", "*Man*", and "*Woman*", the result of "*King*" + "*Man*" - "*Woman*" would be "*Queen*". This means the closest word to the equation is "*Queen*" (Goodman, C. Zimmerman, and Hudson 2020). Figure 4.2 illustrates the five phases of the proposed approach, which can be described as follows:

- The first phase reads each network packet and extracts each n-gram (2 bytes) in hexadecimal format. It counts the appearance frequency of n-grams to create a dictionary. By representing each 2-byte segment (n-gram) of the packet, this phase helps break down the packet's structure into smaller n-grams for analysis.
- The second phase (create vectors) generates integer vectors by applying the dictionary created in the previous phase on the network packets. The conversion to integer vectors helps translate the packet content into a numerical format that is easier for Word2Vec to process.
- The third phase (create word embedding) processes the integer files using the Word2Vec approach to generate word embeddings.

- Next, the results generated from the last two phases, including word embeddings and integer vectors, are used to create feature files.
- Finally, the classification phase runs the training, validation, and inference processes using feature files generated from the DARPA dataset. The authors utilise RF classification to evaluate detection efficiency.

The Packet2Vec approach introduces the technical processes for analysing network packets. The authors evaluate the system based on two metrics: processing time and detection efficiency. The implemented system has a data pre-processing speed of 10.3 MB/s; thus, it handles 559 GB of packet data in approximately 15.5 hours. The detection accuracy and F1-score are 0.63 and 0.751, respectively. Additionally, the approach treats all bytes in the header and payload of packets equally, eliminating the need for a real-time feature extraction module.

The authors in (A. Khan and Cotton 2021) discuss a similar concept about featureless modelling. This concept converts network packets into a byte stream to use as input for ML models. In detail, they propose anomaly detection approaches that do not rely on a feature extraction module. Instead of processing data based on the extracted features, the proposed approaches treat network packet data as a continuous stream of bytes, handling it as a time-series network traffic analysis task. The 1D-CNN models are proposed in this approach to capture sequential patterns in the packet data. Figure 4.3 illustrates the proposed approaches, namely session-based, flow-based, and packet-based. The three approaches follow the same processing flow, which can be explained below:

- The data filtering phase filters packets according to their approach (session-based, flow-based, or packet-based). In other words, this phase rearranges packet order in pcap files so that packets in the same approach are placed next to each other.
- The data pre-processing phase uses network tools to edit each packet following four different scenarios.
- The classification phase includes training, validating, and inference processes using 1D-CNN models.

They have evaluated the proposed approaches on the IoT-23 dataset, focusing on detecting botnet attacks. Most packets related to the port scanning attack

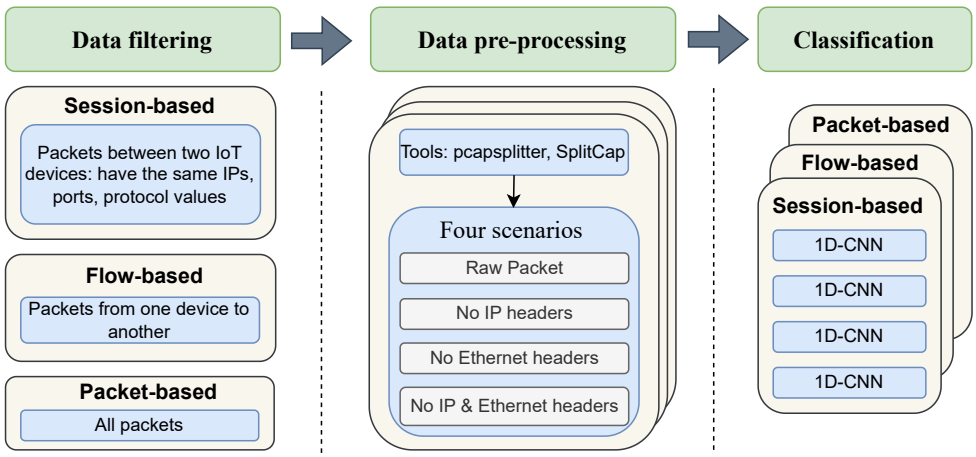


Figure 4.3: The featureless modelling approaches using 1D-CNN.

were removed from this dataset, resulting in a much smaller size dataset. The test results showed that the 1D-CNN model achieved compatible accuracies, with the flow-based method achieving the best results at 99% accuracy and an F1-score of 0.88%. However, when using the flow-based method, the real-time detection model still requires a pre-processing block to filter packets within the same flow before inference.

Overall, the featureless modelling approach is quite interesting since the system doesn't have to categorise packets with different protocols or versions, which reduces complexity in the data pre-processing phase. This approach can also be applied to raw packet data; thus, it has the potential for ML models to learn information from both the headers and the encrypted payload. Additionally, using hardware accelerators to accelerate this approach is promising, as most research has been conducted on software-based platforms, with the contribution lying in the methodologies. The deep packet inspection systems using CNN models will be explored to further experiment with what has been established in the previous chapters.

4.1.2 The IoT-23 Dataset

The IoT-23 dataset (Garcia, Parmisano, and Erquiaga 2020b) contains network traffic captured from IoT devices in the Stratosphere Laboratory. Figure 4.4 shows the labelling process on the IoT-23 dataset. The network packets are fed to the Zeek network analysis framework to generate network flows, which are labelled by the predefined rules in a Python script (Faber). There is a total of

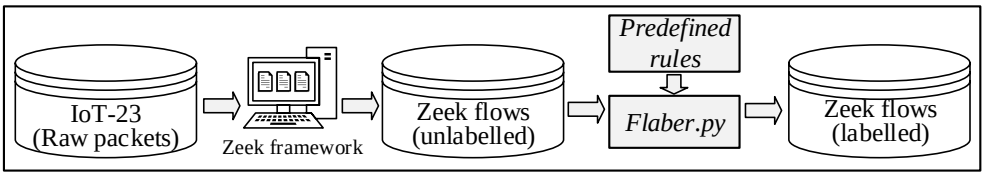


Figure 4.4: The IoT-23 dataset's labelling process flow

325,307,990 labelled flows belonging to sixteen categories.

A flow comprises eighteen features, categorised into four groups, namely static, basic, content, and time features. Static features can be extracted from the packet header, while other features are computed from packets within the same flow (or in a time window).

4.1.3 Related Work

With AI's rapid and robust development, recent research has increasingly focused on techniques such as data mining, ML, and DL for enhancing the security of IoT networks and mitigating potential vulnerabilities (Al-Garadi et al. 2020; Yang et al. 2022). Anomaly-based intrusion detection approaches have been studied that aim to detect unseen attacks in IoT networks (Bhatia et al. 2019; Fahim and Sillitti 2019).

Table 4.1 summarises the state-of-the-art DPI systems. The trend in Deep Packet Inspection (DPI) for network intrusion detection is evolving from traditional methods that rely on network traffic metadata to more advanced techniques that incorporate packet payload data and AI (Farrukh et al. 2023). There's a growing emphasis on analyzing packet payload data using methods like Natural Language Processing (NLP) and image processing. Recent studies use ML to enhance DPI (Zhou et al. 2025; I. Khan, Farrukh, and Wali 2024; Pasquini et al. 2024). In addition, the focus is shifting towards real-time detection by analysing sequential packets and leveraging image-based representations to capture temporal-spatial relationships (Ghadermazi, A. Shah, and Bastian 2024), enabling quicker responses to network attacks. These advancements aim to create more robust and adaptable network IDSs.

Supervised learning techniques have been used to train the IoT-23 dataset in the relevant works. Table 4.2 provides an overview of anomaly-based approaches on the IoT-23 dataset. An ensemble strategy combining DNN and LSTM models has achieved 99.7% accuracy (Dutta et al. 2020a). The authors in (Hussain et

Table 4.1: Summary of state-of-the-art DPI systems.

Work	Dataset	Method	Discussion
(Zhou et al. 2025)	NSL-KDD, UNSW-NB15, AWID2, CICIDS2017 and NF-BoT	Uses a paragraph embedding model to incorporate the protocol hierarchy of attributes, and a synthetic oversampling method to handle class imbalance.	Addresses hierarchical dependency and class imbalance effectively. However, high complexity may lead to computational challenges.
(Pasquini et al. 2024)	UNSW-NB15	Use a 1D CNN and deflate compression algorithm with PCA to encode packets into integer vectors. The system trains three multiclass models to predict the network packets.	Focuses on computational efficiency, suitable for IoT devices. Automated feature extraction minimizes manual intervention.
(Hajaj et al. 2024)	QUIC Davis and FLASH dataset	Proposes a data augmentation technique using LSTM networks to generate synthetic data points.	Enhances dataset diversity and improves classifier performance. However, there are potential complexities in distinguishing genuine traffic from synthetic anomalies.
(Ghadermazi, A. Shah, and Bastian 2024)	CIC-IDS2017 and CIC-IDS2018	Transforms sequential packets into a two-dimensional image and uses a CNN for intrusion detection.	Addresses limitations in packet-based NIDS by considering header and payload data. The work relies on specific datasets, which might limit generalizability.
(Grini et al. 2024)	CIC-IDS2017	Introduces a Hierarchical Packet Attention Convolution System for Intrusion Detection Systems (HPAC-IDS), blending hierarchical attention mechanisms with convolutional neural networks	Employs hierarchical structure for nuanced understanding of packet interdependencies. However, the system is vulnerable to header-based attacks if payload embedding is solely relied upon.
(I. Khan, Farrukh, and Wali 2024)	CIC-IDS2017	Introduces an integrated stacking approach with deep concatenated 2D CNN models. Transforms high-dimensional packet payload data into grayscale images based on byte frequency.	Achieves high F1-score in multiclass classification tasks. However, the work focuses solely on packet-level information, disregarding flow-based data.
(Farrukh et al. 2023)	CIC-IDS2017 and UNSW-NB15	Transforms network traffic into RGB images and uses a deep concatenated CNN model (SeNet-I) for classification.	Capture spatial aspects of network traffic. The system only considers packets' payload data, disregarding the packets' header information.

Table 4.2: Summary of anomaly-based IDS on the IoT-23 dataset

Publication Year	Dataset	Method	IoT-23 Data range
2020 (Dutta et al. 2020a)	IoT-23, LITNET-2020 and NetML-2020	DNN and LSTM	Partial (seven scenarios)
2020 (Hussain et al. 2020)	CICIDS2017, CTU-13 and IoT-23	Naive Bayes (NB), KNN, RF, and Logistic Regression	Mirai botnet attacks
2020 (Hegde et al. 2020)	IoT-23 and Generated dataset	DT, RF, and NN	Partial (botnet attacks)
2021 (A. Khan and Cotton 2021)	IoT-23	1D-CNN	Partial (twelve scenarios)
2022 (Ullah and Mahmoud 2022)	NSL-KDD, BoT-IoT, IoT-NI, IoT-23, MQTT and IoT-DS2	LSTM, and Gated Recurrent Unit (GRU)	Not mentioned
2022 (Nobakht, Javidan, and Pourebrahimi 2022)	IoT-23	1D-CNN	Partial (ten scenarios)
2022 (Alani and Miri 2022)	TON-IoT, IoT-23 and IoT-ID	RF, LR, DT, Gaussian Naive-Bayes (GNB)	Partial (100,000 records each file)

al. 2020) introduce a universal feature set to enhance the effectiveness and accuracy of botnet detection in the IoT-23 dataset. The literature (Ullah and Mahmoud 2022) has pre-processed raw packets using CICFlowmeter to extract and label data for training with DL methods. Nevertheless, it is worth noting that the feature extraction engine proposed in the paper relies on third-party tools, which may introduce additional computational overhead. The malware and botnet behaviour analysis in the IoT-23 dataset has been investigated in (Hegde et al. 2020), and they reported 99.8% accuracy using a NN model. (Nobakht, Javidan, and Pourebrahimi 2022) and (A. Khan and Cotton 2021) have experimented with the one-dimensional CNN technique, and they have achieved a high accuracy of 99% by employing parameter optimisation techniques on a deep ensemble model consisting of CNNs and RF. Similarly, ML algorithms were utilised in (Alani and Miri 2022) to develop lightweight detection models, resulting in an accuracy of 99.61%.

Overall, the relevant works on the IoT-23 dataset typically process labelled Zeek flows, and they utilise a subset of this dataset instead of looking at the entire dataset, due to computational resource limitations. In contrast, this chapter involves processing the raw network packets by labelling each packet in the IoT-23 dataset beforehand; then training them using CNN models. This approach allows a deeper analysis of the network traffic and enables us to capture the characteristics and patterns at the packet level.

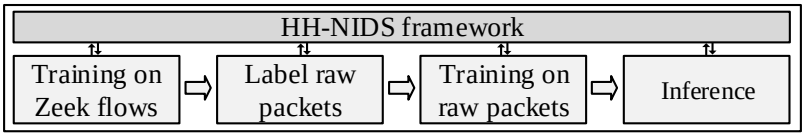


Figure 4.5: The proposed approach's process flow using the HH-NIDS framework (Ngo, Lightbody, Temko, Pham-Quoc, et al. 2023a) on the IoT-23 dataset.

4.2 Proposed Approach

4.2.1 Overview

This chapter utilises ANN and CNN models to detect network attacks by analysing all incoming network packets into the system. The IoT-23 dataset, which is described in Section 4.1.2, is used for the training and inference phases. In the IoT-23 dataset, the labelled data consists of Zeek flows; thus, the proposed approach starts from Zeek flows to generate the ANN models. The trained ANN models can be used to label the raw packets in the IoT-23 dataset; then, the CNN models are trained with the labelled data to produce detection models, which can be used in the inference phase to scan network packets. This approach falls under semi-supervised learning, where ANN models are used to generate labels for unlabelled data, and then the CNN models are trained on the labelled dataset. In this approach, if ANN models make mistakes, CNN models will learn from incorrect data, leading to error propagation. Figure 4.5 illustrates the four main stages in the process flow.

4.2.2 Implementation

HH-NIDS is the framework designed for network intrusion detection in IoT networks. It leverages lightweight neural network models to efficiently detect network attacks. Figure 4.6 illustrates the HH-NIDS's customised version employed in this work. It consists of three main stages: data pre-processing, training, and inference.

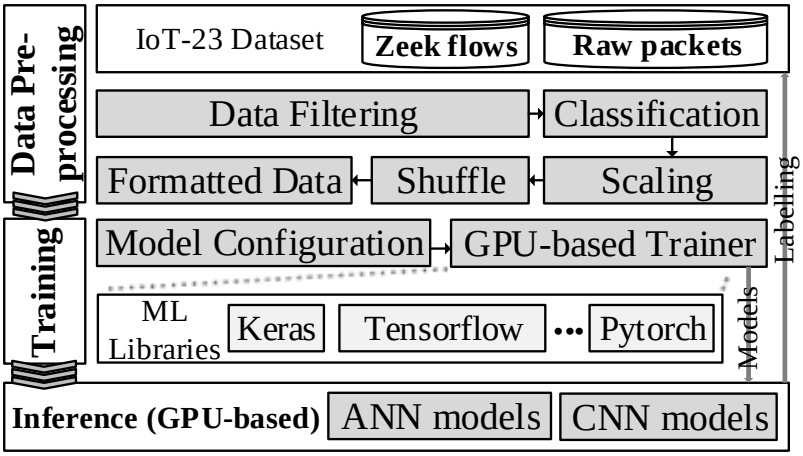


Figure 4.6: HH-NIDS framework architecture

The data pre-processing stage is responsible for extracting and normalising the data from the IoT-23 dataset. The training stage uses the GPU platform to generate machine-learning models based on the labelled data. The inference stage evaluates the generated models on the test dataset. However, the proposed approach involves utilising the generated ANN model to assign labels to the raw packets in the IoT-23 dataset, after which CNN models are constructed based on these raw packets.

Figure 4.7 illustrates the four stages of our proposed approach, namely Training (ANN), Labelling, Training (CNN), and Inference:

Training (ANN) uses the supervised-learning method to generate lightweight ANN models from Zeek flows data. This stage has been adapted from the previous work on the HH-NIDS framework. In the *data pre-processing* phase, the *IP addresses*, *ports*, and *protocol* are extracted and used as inputs for neural network models. The hidden layer of the NN model is configured with 32 neurons, each followed by a Rectified Linear Unit (ReLU) activation function. The dataset proportions for training, validation, and inference are 50%, 25%, and 25%, respectively. The training phase uses the cross-entropy loss function and the Adam optimiser (PyTorch Contributors 2023). In addition, the trainer is configured with a batch size of 512, and the learning rates are 0.0005, 0.001, or 0.01.

Labelling assigns labels to each packet in the IoT-23 dataset using the best ANN model regarding inference accuracy from the first stage. Based on the characteristics of labelled Zeek flows, raw packets need to be filtered before

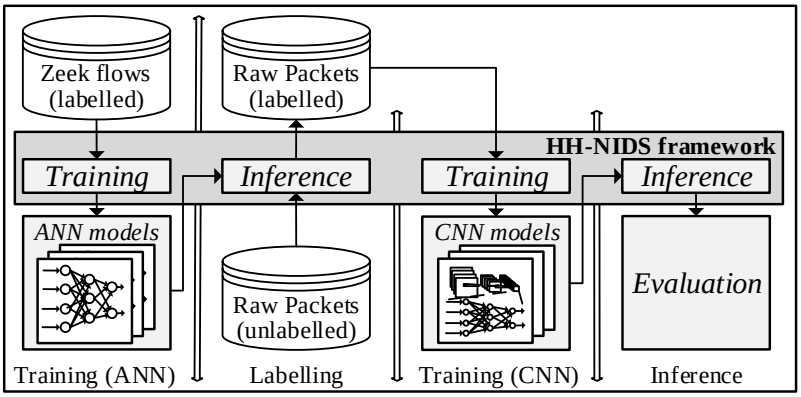


Figure 4.7: The four stages in the process flow, namely Training (ANN), Labelling, Training (CNN), and Inference

labelling. The proposed approach only considers packets that belong to IPv4 (Internet Protocol version 4) and one of the three protocols: TCP, UDP, or ICMP. After applying this filter, there is a total of 813,621,340 packets. Among them, 274,882,686 packets are classified as benign by the ANN network, while 538,738,654 packets are marked as attacks.

Training (CNN) processes labelled packets from the previous stage to generate CNN models. The data pre-processing in the HH-NIDS framework transforms each packet in the dataset into a grey-scale image format with a size of 40x40 pixels (1600 bytes). This size is determined by the maximum size of a packet, known as the Maximum Transmission Unit (MTU). In the Internet's Transmission Control Protocol context, the MTU determines the maximum packet size for any transmission. It is commonly associated with the Ethernet protocol, where a 1500-byte packet is the maximum size allowed (Scott and Josh 2021). Thus, selecting the size of 40x40 pixels ensures the coverage of all packets in the dataset. For packets smaller than this size, padding with zeros is applied.

There are a total of 813,621,340 packets, which are allocated as 80%, 10%, and 10% for training, validation, and inference, respectively. The CNN is also trained with a cross-entropy loss function and the Adam optimiser. The batch size is 512, and the learning rate is 0.001 or 0.01. Table 4.3 presents the CNN model architecture with four 2D-CNN layers. To optimize the CNN model for FPGA deployment, additional layers were incorporated to reduce the overall model size while maintaining performance. This approach minimised the parameter count, making the model more resource-efficient. The design aimed to balance accuracy and resource usage, ensuring feasibility for FPGA implementation without significant performance loss.

Table 4.3: The implemented 2D-CNN model structure

Layer	Operation	Input	Filter	Output
0	Conv2D + ReLU	40*40	3*3	32*40*40
1	MaxPooling	32*40*40	2*2	32*20*20
3	Conv2D + ReLU	32*20*20	3*3	32*20*20
4	Conv2D + ReLU	32*20*20	3*3	32*20*20
5	MaxPooling	32*20*20	2*2	32*10*10
6	Conv2D + ReLU	32*10*10	3*3	32*10*10
7	Flatten	32*10*10	-	3200
8	Fully Connected + ReLU	3200	-	3200*20
9	Fully Connected	20*16	-	2/16

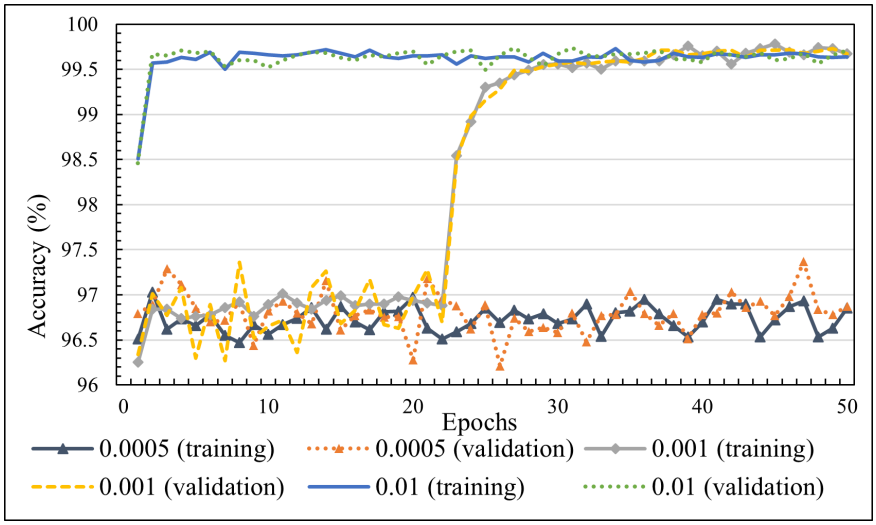
Inference evaluates the trained CNN models on the test dataset. The selected CNN model will be used to classify network packets in real time.

4.3 Experiment

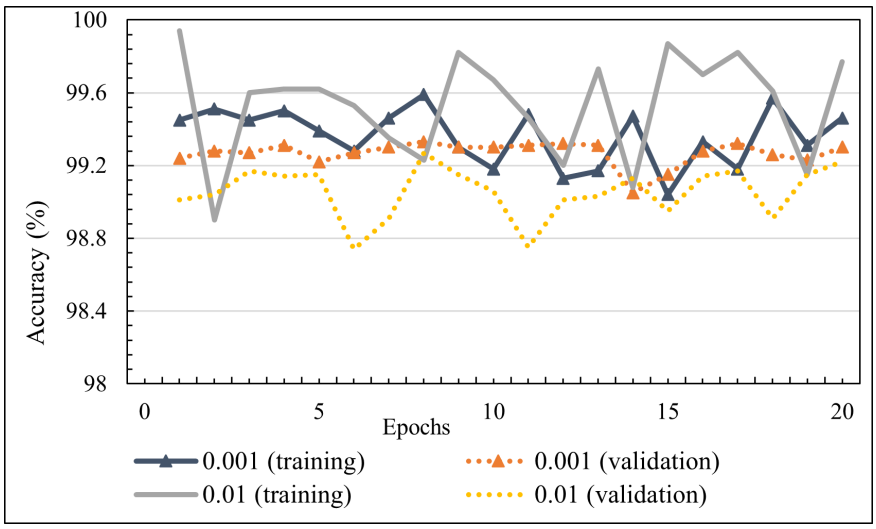
4.3.1 Accuracy

In (Ngo, Lightbody, Temko, Pham-Quoc, et al. 2023a), evaluation metrics, including accuracy, precision, recall, and F1-score, are introduced. Figure 4.8 shows ANN and CNN models' training and validation accuracy on the IoT-23 dataset. The line chart in Figure 4.8a depicts the trained results of ANN models with three different learning rates (LR). The ANN model with an LR of 0.01 achieved a training accuracy of 98.51% in the first epoch, which increased to 99.69% in the subsequent epoch. The accuracy remained relatively stable for the remaining training epochs. Similarly, the ANN model with an LR of 0.001 yielded comparable results, reaching a peak validation accuracy of 97.72% at epoch 45, while the smallest LR (0.0005) can only reach 97.37% accuracy at epoch 46.

The second line chart in Figure 4.8b represents the CNN models' trained results on the labelled packets in the IoT-23 dataset. Due to the larger dataset and model size, the experimented are conducted with two LR values for the CNN and trained for twenty epochs. The results show that the training and validation accuracy values consistently remained between 98.8% and 99.9%. With an LR of 0.01, the accuracy fluctuated in a broader range, achieving better training accuracy. However, validation accuracy is a higher-priority metric used to evaluate a model. In this regard, an LR value of 0.001 yielded superior validation



(**a**) ANN



(**b**) CNN

Figure 4.8: The trained results on the IoT-23 dataset with different learning rates

accuracy, reaching 99.33% at epoch eight.

The next experiment compares the inference accuracy of the proposed approach with related work. To obtain the inference accuracy values, the selected the models with the best validation accuracy during the training process and used these models on the unseen dataset.

Table 4.4 shows the inference accuracy and F1-score values compared to related work. By applying DNN and LSTM, the authors in (Dutta et al. 2020a) and (Ullah and Mahmoud 2022) have reported 99.70% and 99.83% accuracy, respectively. In (Hussain et al. 2020), the authors have claimed that RF pro-

Table 4.4: Inference accuracy and F1-score comparison to related work

System	Method	Accuracy (%)	F1-score (%)
(Dutta et al. 2020a)	DNN and LSTM	99.70	98.00
(Hussain et al. 2020)	RF	100.00	100.00
(Hegde et al. 2020)	NN	99.80	99.85
(A. Khan and Cotton 2021)	1D-CNN	100.0	97.00
(Ullah and Mahmoud 2022)	BiLSTM	99.83	99.99
(Nobakht, Javidan, and Pourebrahimi 2022)	1D-CNN (Ensemble)	99.90	99.90
(Alani and Miri 2022)	DT	99.61	99.61
Proposed system	ANN	99.71	99.84
	CNN	99.34	99.51

duced the highest accuracy and an F1-score of 100% for detecting attacks on the selected scenarios. Researchers in (Hegde et al. 2020) have achieved 99.8% accuracy and 99.85% F1-score with ANN models. 1D-CNN is employed in (A. Khan and Cotton 2021) and (Nobakht, Javidan, and Pourebrahimi 2022), both achieved higher 99.9% accuracy and the F1-score are 97% and 99.9%, respectively. Finally, authors in (Alani and Miri 2022) with DT technique have recorded 99.61% in accuracy and F1-score. However, these experiments have not been applied to the full IoT-23 dataset, as mentioned in Section 4.1.3.

The proposed approach involved working with the full IoT-23 dataset. Pre-trained ANN models from the previous work (Ngo, Lightbody, Temko, Pham-Quoc, et al. 2023a) (HH-NIDS framework) have achieved excellent results, with an accuracy of 99.71% and an F1-score of 99.51% for binary classification. These models are utilised to label each packet, resulting in a dataset that is 2.5 times larger than the provided data (Zeek flows). The CNN model is constructed using this dataset, which recorded accuracy and F1-scores of 99.34% and 99.51%, respectively.

4.3.2 Performance

The proposed approach was implemented on the Dell Precision 7920 Tower with 512GB of memory and Intel®Xeon(R) Platinum 8276 CPU @ 2.2GHz x 112 running Ubuntu 22.04.2 LTS operating system. The NVIDIA Quadro GV100 GPU with 32GB memory was the hardware accelerator for training and inference.

The fetching input data is configured with 32 processors, and the NVIDIA Quadro GV100 GPU is used for training; the average training time (one epoch) of ANN and CNN models are 7.7 and 229.9 minutes, respectively. Next, the performance evaluation of the system using network packets is conducted.

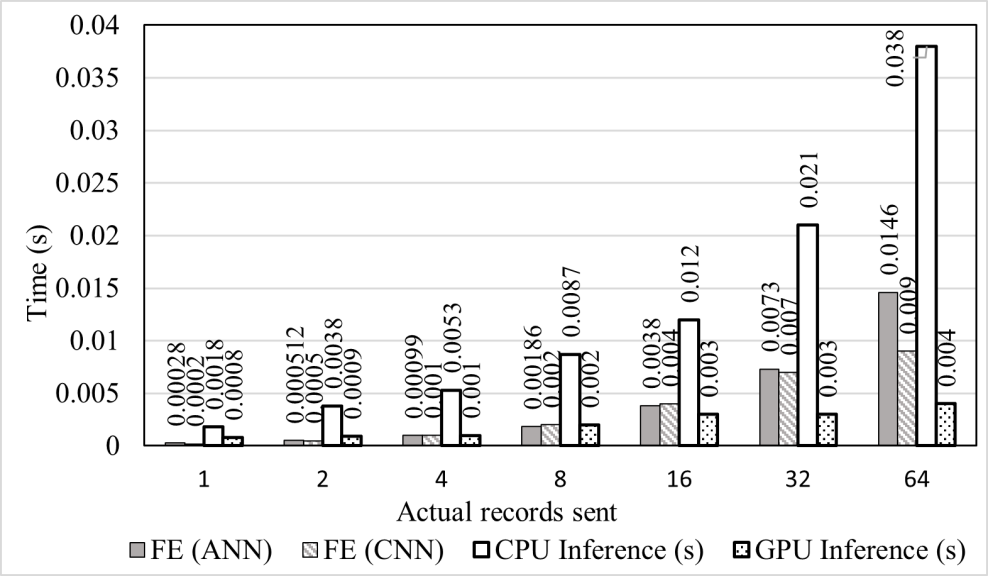


Figure 4.9: Feature extraction and inference time on CPU and GPU for different input buffer sizes.

The different numbers of packets are sent to the system to measure the execution time for feature extraction (FE) and inference time on both CPU and GPU platforms. The bar chart in Figure 4.9 shows the performance results (processing time) between CPU and GPU during inference. The horizontal and vertical axes present the number of actual records sent per test and the execution time, respectively.

At the beginning of the test, with only one record sent, the average FE time is 0.00028s and 0.0002s on CPU and GPU, respectively. By doubling the number of records sent each time, the FE time and inference time on CPUs are likely to be doubled; these numbers reached 0.0146s and 0.038s when grouping 64 inputs. The GPU reported better results in both FE time and inference time, which increased slowly to 0.009s and 0.004s, respectively, at the end of the test. If each packet is extracted and sent immediately to the system, the CNN models show an approximately 40% improvement in feature extraction time compared to ANN models.

4.4 Summary

This chapter introduced an approach for detecting network attacks on IoT devices using 2D-CNN models. The proposed approach involves labelling network packets in the IoT-23 dataset from pre-trained ANN models; then, 2D-CNN mod-

els are trained to detect attacks at the packet level.

The novelty of this approach lies in the fact that the CNN model considers each byte of data in the packet equally, which differs from the ANN model. In an ANN model, a pre-processing step is required to extract input features for the model to learn and produce decisions. By utilising the CNN model, all data in the packet is treated as valuable in contributing to the final decision. However, this also implies that accurate labelling of each packet is crucial during training processes.

The results show that ANN and 2D-CNN models have achieved accuracies of 99.71% and 99.34% on the IoT-23 datasets, respectively. Moreover, the proposed approach with 2D-CNN mitigates the computational overhead of the feature extraction process associated with the ANN, resulting in an approximately 40% improvement in feature extraction time. The CNN models are optimised for implementation on FPGA in the next chapters.

4.4.1 Optimise CNN models

From the beginning, the proposed approach intends to accelerate CNN models on FPGA hardware, especially using network-specialised hardware for studying high-performance network attack detection systems. However, when implementing ML models on FPGA hardware, one must be aware of the model's size, as the hardware resources are limited. Optimising ML models and parameter quantisation techniques can reduce the ML model's size.

- Optimising the ML model is the process of exploring different model structures to find smaller models while maintaining an acceptable level of accuracy.
- The Quantisation process adjusts the model's parameters to fit within a specific range. Additionally, it involves converting these parameters from floating-point numbers to integers, which helps reduce the computational complexity and memory requirements when deploying the model on hardware.

Table 4.5 presents different CNN models and their training and validation accuracy results.

Table 4.6: The CNN model structure in the fifth trial.

Layer	Operation	Input	Filter	Output
0	Conv2D + ReLU	40*40	3*3	8*40*40
1	MaxPooling	8*40*40	2*2	8*20*20
3	Conv2D + ReLU	8*20*20	3*3	8*20*20
4	MaxPooling	8*20*20	2*2	8*10*10
5	Conv2D + ReLU	8*10*10	3*3	8*10*10
6	MaxPooling	8*10*10	2*2	8*5*5
7	Conv2D + ReLU	8*5*5	3*3	8*5*5
8	MaxPooling	8*2*2	2*2	8*2*2
9	Flatten	8*2*2	-	32
10	Fully Connected + ReLU	32	-	32*20
11	Fully Connected	20*16	-	2/16

Table 4.5: The training and validation results of different CNN models (C: Convolution layer, P: Pooling layer, D: Dense layer).

Model	Channel	Structure	Training (%)	Validation (%)
1	32	CP-C-CP-C-DD	99.34	99.51
2	16	CP-CP-CP-C-DD	99.89	99.46
3	8	CP-CP-CP-C-DD	99.19	98.99
4	4	CP-CP-CP-C-DD	90.27	81.61
5	8	CP-CP-CP-CP-DD	98.60	98.09

A testing strategy was employed in which the number of channels was gradually reduced, and additional convolution and pooling layers were inserted to reduce the overall model size. In the first three trials, the approach successfully reduced the number of channels from 32 to 16, and then to 8, with a slight decline in accuracy, from 99.51% to 99.46% and 98.99%, respectively. However, in the fourth trial, when the number of channels was reduced to 4, accuracy dropped significantly to 81.61%.

The final trial increased the number of channels back to eight and inserted an additional max pooling layer at the end to further reduce the size of the CNN model by half. The results showed that accuracy increased to 98.09%. Table 4.6 presents the CNN model architecture with four convolution and pooling layers. This model architecture was selected for implementation on FPGA hardware in Chapter 6.

From the perspective of this chapter, the CNN model detects attacks using both header fields and packet data, while the ANN model only focuses on header

fields. Future work aims to develop a dataset with fully labelled network packets to train the CNN model. For instance, the power data in (Lightbody et al. 2023) can be accelerated with 2D-CNN models to build a system for detecting and classifying attacks.

The next chapter aims to explore security at the Cloud level. Specifically, distributed-ledger technology, Blockchain, and smart contracts for IoT security are focused. The secure framework has been established at the edge level; developing a security solution in the cloud that emphasises flexibility and scalability is crucial, especially when there is always a demand for system expansion. Distributed-ledger technology has enormous advantages in this context, as it provides an immutable environment, ensures data integrity, and is highly scalable.

References

Alani, Mohammed M and Ali Miri (2022). "Towards an Explainable Universal Feature Set for IoT Intrusion Detection". In: *Sensors* 22.15, p. 5690.

Bhatia, Randeep et al. (2019). "Unsupervised machine learning for network-centric anomaly detection in IoT". In: *Proceedings of the 3rd ACM CoNEXT workshop on Big Data, Machine Learning and Artificial Intelligence for Data Communication Networks*, pp. 42–48.

Cybersecurity, Canadian Institute for (2016). *ISCX VPN-nonVPN Traffic Dataset*. Accessed: Dec 16, 2024. URL: `https://www.unb.ca/cic/datasets/vpn.html`.

Dainotti, Alberto, Antonio Pescape, and Kimberly C Claffy (2012). "Issues and future directions in traffic classification". In: *IEEE network* 26.1, pp. 35–40.

Dutta, Vibekananda et al. (2020a). "A deep learning ensemble for network anomaly and cyber-attack detection". In: *Sensors* 20.16, p. 4583.

Fahim, Muhammad and Alberto Sillitti (2019). "Anomaly detection, analysis and prediction techniques in IoT environment: A systematic literature review". In: *IEEE Access* 7, pp. 81664–81681.

Farrukh, Yasir Ali et al. (2023). "Senet-i: An approach for detecting network intrusions through serialized network traffic images". In: *Engineering Applications of Artificial Intelligence* 126, p. 107169.

Al-Garadi, Mohammed Ali et al. (2020). "A survey of machine and deep learning methods for internet of things (IoT) security". In: *IEEE Communications Surveys & Tutorials* 22.3, pp. 1646–1685.

Garcia, S, A Parmisano, and MJ Erquiaga (2020b). "IoT-23: A labeled dataset with malicious and benign IoT network traffic". In: *Stratosphere Lab., Praha, Czech Republic, Tech. Rep*.

Ghadermazi, Jalal, Ankit Shah, and Nathaniel D Bastian (2024). "Towards real-time network intrusion detection with image-based sequential packets representation". In: *IEEE Transactions on Big Data*.

Gharaibeh, Manaf and Christos Papadopoulos (2014). "Darpa-2009 intrusion detection dataset report". In: *Tech. Rep*.

Goodman, Eric L, Chase Zimmerman, and Corey Hudson (2020). "Packet2Vec: Utilizing Word2Vec for Feature Extraction in Packet Data". In: *arXiv preprint arXiv:2004.14477*.

Grini, Anass et al. (2024). "HPAC-IDS: A Hierarchical Packet Attention Convolution for Intrusion Detection System". In: *2024 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, pp. 1–6.

Hajaj, Chen et al. (2024). "The art of time-bending: Data augmentation and early prediction for efficient traffic classification". In: *Expert Systems with Applications* 252, p. 124166.

Hegde, Mandira et al. (2020). "Identification of Botnet Activity in IoT Network Traffic Using Machine Learning". In: *2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*. IEEE, pp. 21–27.

Hussain, Faisal et al. (2020). "Towards a Universal Features Set for IoT Botnet Attacks Detection". In: *arXiv preprint arXiv:2012.00463*.

Khan, Arshiya and Chase Cotton (2021). "Detecting attacks on IoT devices using featureless 1D-CNN". In: *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, pp. 461–466.

Khan, Irfan, Yasir Ali Farrukh, and Syed Wali (2024). "ByteStack-ID: Integrated Stacked Model Leveraging Payload Byte Frequency for Grayscale Image-based Network Intrusion Detection". In: *ICC 2024-IEEE International Conference on Communications*. IEEE, pp. 2731–2736.

Lightbody, Dominic et al. (2023). "Attacks on IoT: Side-Channel Power Acquisition Framework for Intrusion Detection". In: *Future Internet* 15.5, p. 187.

Lotfollahi, Mohammad et al. (2020). "Deep packet: A novel approach for encrypted traffic classification using deep learning". In: *Soft Computing* 24.3, pp. 1999–2012.

Mikolov, Tomas (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Ngo, Duc-Minh, Dominic Lightbody, Andriy Temko, Cuong Pham-Quoc, et al. (2023a). "HH-NIDS: Heterogeneous Hardware-Based Network Intrusion Detection Framework for IoT Security". In: *Future Internet* 15.1, p. 9.

Nobakht, Mehrnoosh, Reza Javidan, and Alireza Pourebrahimi (2022). "DEMD-IoT: a deep ensemble model for IoT malware detection using CNNs and network traffic". In: *Evolving Systems*, pp. 1–17.

Pasquini, Aleksandar et al. (2024). "Robust and Lightweight Modeling of IoT Network Behaviors From Raw Traffic Packets". In: *IEEE Transactions on Machine Learning in Communications and Networking*.

PyTorch Contributors (2023). *ADAM*. `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html` visited on Dec 16, 2024.

Qi, Yaxuan et al. (2009). "Packet classification algorithms: From theory to practice". In: *IEEE INFOCOM 2009*. IEEE, pp. 648–656.

Scott, Hogg and Fruhlinger Josh (2021). *MTU size issues, fragmentation, and jumbo frames*. `https://www.networkworld.com/article/2224654/mtu-size-issues.html` visited on Dec 16, 2024.

Sherry, Justine et al. (2015). "Blindbox: Deep packet inspection over encrypted traffic". In: *Proceedings of the 2015 ACM conference on special interest group on data communication*, pp. 213–226.

Ullah, Imtiaz and Qusay H Mahmoud (2022). "Design and Development of RNN Anomaly Detection Model for IoT Networks". In: *IEEE Access* 10, pp. 62722–62750.

Velan, Petr et al. (2015). "A survey of methods for encrypted traffic classification and analysis". In: *International Journal of Network Management* 25.5, pp. 355–374.

Yang, Zhen et al. (2022). "A systematic literature review of methods and datasets for anomaly-based network intrusion detection". In: *Computers & Security*, p. 102675.

Zhou, Weidong et al. (2025). "Hidim: A novel framework of network intrusion detection for hierarchical dependency and class imbalance". In: *Computers & Security* 148, p. 104155.

Chapter 5

Distributed-ledger Technology: Blockchain and Smart Contracts for IoT Security

In 2024, the financial market witnessed a new wave with the spot ETF (Exchange-Traded Fund) approvals (Gary Gensler 2024) of the two most popular cryptocurrencies, Bitcoin and Ethereum. These approvals have made cryptocurrencies becoming more valuable and accessible than before. Blockchain, the underlying technology behind cryptocurrencies, has gained significant adoption with numerous emerging applications. Specifically, Blockchain is a form of DLT (Distributed Ledger Technology) that operates a decentralised, peer-to-peer network to maintain synchronised copies of the ledger. This distributed data storage ensures data integrity, redundancy and security by eliminating the single point of failure in the centralised storage systems. Smart contracts were introduced in modern Blockchain networks, which can integrate business logic into the Blockchain. Smart contracts are triggered by transactions and execute automatically without a central entity. This work utilises smart contracts to manage access control in IoT networks.

This chapter studies Distributed-ledger Technology (DLT), Blockchain, and smart contracts. These technologies form the backbone of modern decentralised systems, with DLT and Blockchain offering secure and immutable data records across the network. Next, the leading smart contract technologies that have gained global traction are investigated, including solidity, chaincode, anchor and move. Their architectures and applications in various domains,

from finance to supply chain management, are discussed.

Next, the chapter explores the role of smart contracts in managing and securing IoT devices. Smart contracts can enhance IoT security by automating access control and ensuring data integrity. The focus is on enhancing the scalability and accessibility of IoT networks using smart contracts. Scalability is an important aspect of IoT systems due to the massive number of interconnected devices and the high volume of data generated.

Finally, the chapter proposes smart contracts on the Blockchain to enhance security and scalability in IoT networks. Smart contracts are employed for IoT authentication and authorisation. The proposed approach is evaluated to demonstrate its scalability and security. The data within the system are managed by the smart contract on the Blockchain, which is transparent and immutable.

5.1 Distributed Ledger Technology: Platforms and Applications

DLT (NEVIL 2024) operates based on a digital ledger system with programmed protocols that allow reading, writing and updating the distributed ledger. The idea behind DLT is to maintain identical copies of data in a network of nodes. The nodes communicate with each other through the peer-to-peer principle, where each node maintains the up-to-date version of the ledger. This mechanism provides transparency, resilience and security, as there is no single point of failure compared to traditional centralised systems. Thanks to these advantages, DLT has become increasingly popular with broad applications across various sectors. The fundamental technologies behind DLT involve cryptographic techniques, such as hash functions, digital signatures and public key cryptography. Besides, this is the technology behind Blockchain, directed acyclic graphs (DAGs), hashgraphs, corda, bigchainDB, etc.

5.1.1 Blockchain and Consensus Mechanisms

Blockchain (Jha 2023) is the most successful technology under DLT due to the ongoing adoption of Bitcoin, the oldest cryptocurrency. Blockchain forms a chain of blocks, where the current block is built based on the previous block's header hash value and the combination of transactions, a timestamp, a nonce and a Merkle root to include a hash of all transactions. Blockchain inherits core

characteristics from DLT, such as data integrity, immutability and redundancy. Blockchain can be categorised based on the permission model and consensus mechanism used. There are three Blockchain types according to permission models, as follows:

- *Public Blockchains (Permissionless)* are distributed computer networks open to the public. Any nodes that satisfy certain requirements in computing resources and network bandwidth can join the system. Clients can request or validate transactions publicly on the Blockchain. A public Blockchain is fully decentralised, as there is no central or third-party authority involved.
- *Private Blockchains (Permissioned)* are not open to the public since they have access restrictions. Clients need to acquire permissions before interacting with the Blockchain. A private Blockchain is dedicated to a specific group of users or an organisation; in other words, it is centralised to some extent.
- *Hybrid and Consortium Blockchains* combine the features of both private and public Blockchains. While a hybrid Blockchain is usually controlled by one authority, a consortium Blockchain is controlled by a group of organisations rather than a centralised entity.

A **consensus** mechanism is used in a Blockchain network to achieve agreement between nodes (validators) before adding a new block to the Blockchain. It can be used as a criterion to distinguish between Blockchains. This section discusses the two most popular consensus mechanisms, which are PoW (Proof of Work) and PoS (Proof of Stake).

- PoW consensus was originally used in the Bitcoin and Ethereum Blockchains. Nodes in the PoW network try to solve a cryptographic puzzle, which represents the hashed value in the new block to be added to the Blockchain. The main concept in PoW is that the puzzle is difficult to solve but easy to verify; when a node finds a possible result, it broadcasts to the network and the block is added when more than 50% of nodes agree with the result. Blockchains using PoW are secured since the attackers need to control more than 50% of the votes to add a malicious block.
- PoS consensus was proposed to reduce the computational resources in PoW. Generally, in a PoS network, each node has to lock up the native

asset (cryptocurrency) to join the consensus protocol, in which the higher the number of asset units staked, the higher the probability the node will be chosen to add the next block.

Besides PoW and PoS, there are other consensus mechanisms, such as Delegated Proof of Stake (DPoS), Proof of Authority (PoA), Practical Byzantine Fault Tolerance (PBFT), Proof of Elapsed Time (PoET), Proof of Burn (PoB), Nominated Proof of Stake (NPoS), Proof of History (PoH), etc. Each consensus has different advantages; thus, it can be selected for a Blockchain depending on design choices. Table 5.1 shows a summary of consensus mechanism in different Blockchain platforms.

5.1.2 Bitcoin Blockchain: Structure and Operations

As discussed in the previous section, a Blockchain operates on the core concept that blocks, representing the generated data over time, are linked together to form a chain. This chain is stored in many Blockchain nodes to ensure reliability, redundancy, and immutability. Although Blockchains can be different from each other depending on their purposes, there is no denying that the Bitcoin Blockchain is the most successful and has brought significant recognition to Blockchain technology in general. This section explores the structure and operations of the Bitcoin Blockchain.

Bitcoin (BTC) was first introduced as an open-source, peer-to-peer currency (Nakamoto 2009). Figure 5.1 illustrates the Bitcoin Blockchain structure with the block header's information.

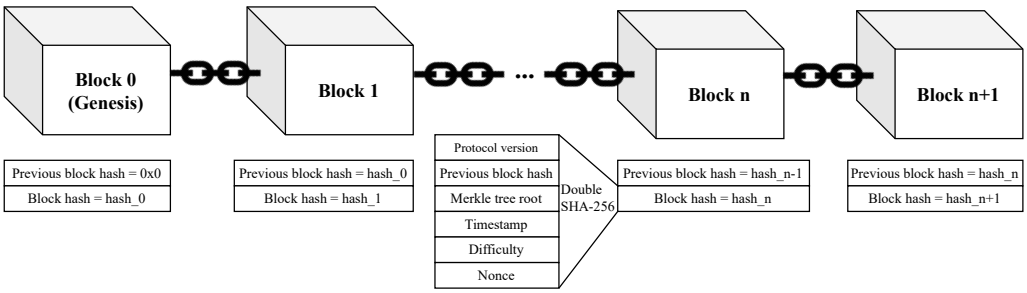


Figure 5.1: Bitcoin Blockchain using double SHA-256 hashing technique to generate the block hash.

A block is minted when miners generate and agree on a hash value, which is generated using double SHA-256 on the block header fields (protocol version, previous block hash, Merkle root hash, timestamp, difficulty, and nonce). The

Table 5.1: Summary of popular consensus mechanisms in the Blockchain platforms.

Consensus	Description	Applications	Blockchains
PoW	All nodes compete to solve cryptographic puzzles to create a new block.	Ideal for decentralized, permissionless Blockchains with high-security requirements while sacrificing computational power from all nodes.	Bitcoin, EthereumPoW, Litecoin.
PoS	Nodes have to lock a certain amount of the chain's native asset to join the consensus.	Reducing energy consumption, with better transaction speeds and low fees compared to PoW.	Ethereum, Cardano, Algorand Foundation.
DPoS	Trusted nodes are voted and promoted to constitute a validator group.	Suitable for semi-decentralized platforms with higher throughput.	EOS, TRON.
PoA	Nodes are pre-approved based on their identity and reputation.	Ideal for private Blockchains focused on high speed, security, and restricted access control.	Hyperledger Fabric.
PBFT	Consensus is reached through voting rounds among a group of approved nodes in the Blockchain.	Ideal for private Blockchains that prefer speed and fault tolerance.	Hyperledger Fabric, Ripple.
PoET	Requires a trusted execution environment to generate a wait time randomly. The node with the shortest wait time can propose a block.	Suitable for private Blockchains, often used in consortium Blockchain settings for resource and energy efficiency.	Hyperledger Sawtooth.
NPoS	A node is selected through a nomination and staking process (similar to PoS with an added secure layer).	Used in networks focused on decentralized governance and security.	Polkadot.
PoH	Nodes use verifiable delay functions to create a historical record proving that events occurred in a specific sequence.	Optimised for high throughput permissionless Blockchains.	Solana.
AptosBFT	Uses Byzantine Fault Tolerant consensus that enables parallel execution.	Suitable for scalable Blockchains with high throughput needs using parallel processing.	Aptos.

link between hashes' values presents the Bitcoin Blockchain, in which the next block has the previous block hash in its block header.

Genesis block (block 0) is the first block in the Bitcoin Blockchain. It does not have a previous block, so the previous block hash field is hardcoded to zero. Besides block headers, transactions are included in each block. A transaction is the basic element in the Bitcoin Blockchain. They are grouped and broadcast over the network. The miner nodes then validate these transactions and add them to a block. Bitcoin Blockchain's state is defined using the Unspent Transaction Output (UTXO) concept, which presents how much of a particular address

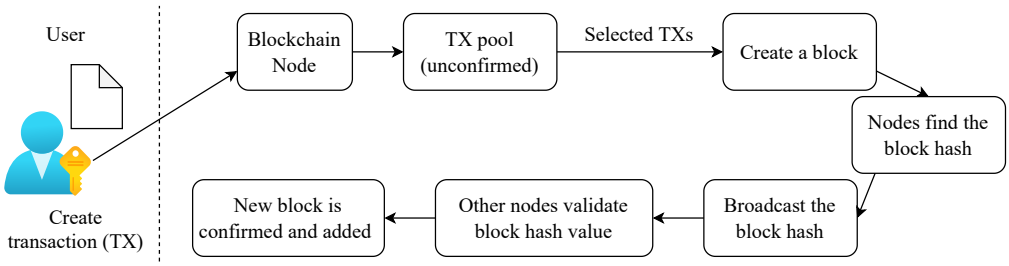


Figure 5.2: Bitcoin Blockchain operation from the user point of view.

balance remains after performing a transaction (sending assessments and paying the fee). UTXO structure includes the transaction's identifier, an index, the spending amount, and the conditional script for spending that amount of BTC. The Block creation process in Bitcoin Blockchain is described in Figure 5.2.

Merkle tree presents a tree data structure containing hashed values. The structure is organised in such a way that it presents all of the data in a file. In Blockchains, the Merkle tree is used to encode transactions in a block efficiently and securely. All transactions are hashed into a value called the Merkle root, which also contributes to creating the block hash. In Bitcoin Blockchain, nodes do not hash all the transactions at once; they concatenate and hash two transactions together, and so on, until there is one hash for the entire block. The Merkle tree allows Blockchain users to verify a specific transaction without the knowledge of an entire Blockchain. It is an efficient data structure in Blockchain for verifying transactions and creating a chain of blocks.

5.1.3 Ethereum Blockchain: Smart Contracts

Ethereum Blockchain is the second most used Blockchain behind Bitcoin Blockchain. When the network was first launched, it originated similarly to the Bitcoin Blockchain, using the PoW consensus mechanism. However, with continuous development and adoption over time, Ethereum has made significant improvements through hard-fork updates to address and eliminate security vulnerabilities. Notably, Ethereum underwent a hard fork [1], namely "The Merge" (Ethereum Foundation 2022), to change from the PoW to the PoS consensus mechanism, reducing transaction costs and further encouraging the

[1] A hard fork in blockchain is a major, incompatible change to the protocol that causes a permanent split in the network. Nodes running the old protocol cannot validate blocks created by nodes running the new protocol, resulting in two separate chains. This usually occurs due to disagreements over changes in protocol rules, such as block size or consensus mechanisms.

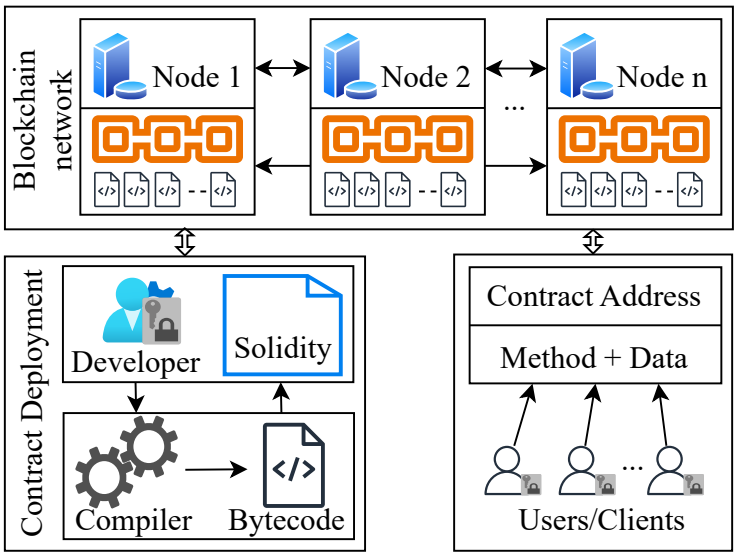


Figure 5.3: The smart contract development and interaction process flow.

development of smart contracts on its network. This section employs smart contract technology based on an Ethereum simulation platform for IoT security.

Smart contract (Ethereum 2024) was first introduced in the 1990s. The smart contract carries the concept of self-executing logic without intermediaries to achieve automation and help reduce enforcement costs. In 2015, Ethereum was launched as the first Blockchain to support complex smart contracts. In the Ethereum Blockchain, smart contracts are self-executing byte codes containing defined logic used to do a specific task. Most importantly, all the changes are recorded on the Blockchain. Thus, historical records remain transparent and immutable. Figure 5.3 illustrates the contract deployment and users' interaction flows on a public Blockchain network. The contract deployment starts with a developer implementing the smart contract using a high-level language, such as Solidity (Solidity Team 2024). The contract is tested and compiled to generate byte code. Next, the contract deployer can deploy the smart contract using this byte code by initiating a contract creation transaction and sending it to the Blockchain network, where miner nodes insert the transaction into a new block and add it to the Blockchain. Clients can interact with the contract using the deployed contract address with the associated method and input data. This process is called interacting with a decentralised application.

In general, **Decentralised Applications** (Dapp) indicate software programs that utilise smart contracts for data management, thus requiring Blockchain

services and Blockchain infrastructure to realise their purpose. A Dapp provides Blockchain access to applications which do not require knowledge about others to perform peer-to-peer transactions.

This work explores the EVM (Ethereum Virtual Machine), an engine of the Ethereum Blockchain. EVM serves as a runtime environment for executing smart contracts on Ethereum. The Ganache (TRUFFLE SUITE 2024) software was used for simulating the EVM environment and also for testing smart contracts. Ganache allows developers to deploy smart contracts in a simulated Blockchain, which reduces cost compared to the live Ethereum Blockchain.

5.1.4 Hyperledger Blockchain

Hyperledger is an open-source, permissioned Blockchain that was developed by Linux Foundation (Linux Foundation 2016). From the start, Hyperledger has been adopted by industry leaders, including IBM and Intel. Unlike Blockchain platforms that use digital assets as an incentive model, Hyperledger aims to build Blockchain frameworks without using digital assets. Compared to public Blockchains like Bitcoin and Ethereum, Hyperledger is designed for the private sectors; thus, it is suitable for smaller-scale environments and organizations prioritising security, privacy, and performance. Figure 5.4 provides a brief overview of Hyperledger Blockchain's evolution stages.

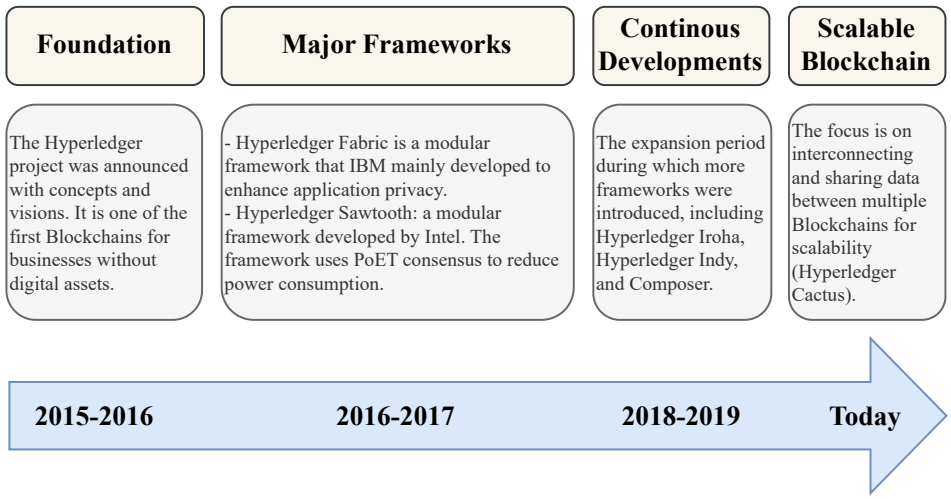


Figure 5.4: Hyperledger Blockchain frameworks' evolution.

Hyperledger Fabric (Hyperledger Foundation 2024a) and Sawtooth are the two

major frameworks developed in 2016. They both focused on the modular frameworks but using different consensus mechanisms. Hyperledger Fabric by IBM can be configured with Practical Byzantine Fault Tolerance (PBFT) consensus, which is well-known for security and resilience. Hyperledger Sawtooth, which is adopted by Intel, uses the Proof of Elapsed Time (PoET) consensus mechanism to reduce the mining cost compared to PoW consensus.

The developments of Hyperledger Blockchain continue to grow until today, with a focus on enhancing scalability and security. The Hyperledger ecosystem is now part of Linux Foundation Decentralized Trust - the flagship organization for fostering innovation and collaboration in the development and deployment of decentralized systems and technologies. Overall, private Blockchains like Hyperledger are suitable for organizations that need full control over the data with different security and privacy levels.

5.1.5 Decentralised Applications: State-of-the-art Platforms

The logic behind Dapps is deployed on a Blockchain through smart contract technology. This section studies the most widely used smart contract technologies, with the vast majority of them operating with reward-incentive Blockchains based on digital assets (or cryptocurrencies). Table 5.2 briefly describes these smart contract technologies, including Solidity, Chaincode (Hyperledger Foundation 2024b), Anchor (Solana Foundation 2024), and Move (Github 2024).

When discussing smart contract platforms and technologies, the focus is on their security, scalability, accessibility, and throughput (TPS - transactions per second). These factors are important in determining how well a platform can support decentralized applications (DApps) and Blockchains.

Table 5.2: Smart contract technologies and platforms.

Technology	**Description**	**Pros and Cons**	**Blockchains**
Solidity	The most used language for developing smart contracts and Dapps on EVM-compatible Blockchains. It is widely adopted by the first generation of smart contract platforms, including a huge number of Ethereum Blockchain variants.	**Pros**: Solidity is well-documented and has rich development tools and community support. **Cons**: The language has faced security vulnerabilities in the past due to its complexity.	Ethereum, EVM-comp (Fantom, Avalanche, Polygon, etc.), Layer-2 Ethereum (Base, Optimism, Arbitrum, etc.).
Chaincode	Introduced by the Linux Foundation as the smart contract technology behind Hyperledger Blockchain. It is accessible by supporting Go, Java and JavaScript for development.	**Pros**: Highly accessible and providing modularity. **Cons**: Come with a complex transaction flow and higher security management requirements.	Hyperledger Fabric
Anchor	Uses a Rust-based framework, aiming at fast transactions. Rust is the primary language used for the development of the smart contract on Solana. It integrates the Solana Program Library (SPL) to include standard programs and utilities for supporting developers.	**Pros**: High-speed transactions with a standardized development process. **Cons**: Limited to Solana with high development complexity and requires an understanding of Rust programming.	Solana
Move	The Move language was developed by Facebook's Libra (Diem) project to focus on security, scalability, and transaction management. It provides custom, resource-oriented programming to control resource ownership and usage.	**Pros**: High level of Security through resource management. High flexibility and adaptability. **Cons**: Small community, limited adoption outside of specific platforms. Requires an understanding of resource-oriented concepts.	Aptos, Sui

Solidity uses formal verification and audit services for security, while Hyperledger is secured by using its private environments. Besides, Scalability is crucial for handling high transaction volumes. Anchor technology in Solana claims a high throughput of 65,000 TPS; this number is over 160,000 TPS in Aptos (Move technology) with the parallel execution engine (Crypto Atlas 2024).

5.2 Blockchain and IoT Security

5.2.1 Blockchain and Anomaly Detection

In (Sharadqh et al. 2023), the authors propose a **security system for an IoT environment** with the primary objective of providing security by implementing an IDS that ensures low computation time and high accuracy. The authors propose security measures across different layers in an IoT environment, as shown in Figure 5.5. At the cloud layer, they suggest utilising a Hybrid chain and smart contracts for user management through two mechanisms. The first mechanism employs a time-based authentication method to encrypt user information and store it in the hybrid chain. The second mechanism uses smart contracts to manage users when they request data from the upper layer, with these requests requiring approval by the smart contract before any executions. Between the device layer and the Blockchain, Edge Servers are integrated with IDS to scan network traffic and detect attacks. In this layer, they use machine learning algorithms, specifically CapsNet and ResNet neural networks.

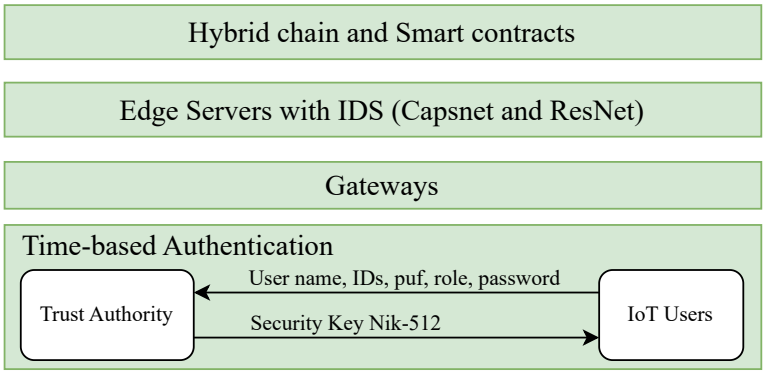


Figure 5.5: Architecture of the proposed HybridChain framework (Sharadqh et al. 2023).

The experimental results were conducted through simulations on the NS3 software (Consortium 2024); the setup consists of 100 users, 4 Gateways, 4 Edge Servers, and 1 Cloud Server hosting the Blockchain. The system was compared with two related works, Lit-IDS (X.-H. Nguyen et al. 2022) and Passban-IDS (Eskandari et al. 2020), in terms of accuracy, detection rate, false alarm rate, and F-score, demonstrating superior results. The research's strength lies in its proposal of a multi-layered security approach for IoT environments. However, this approach requires further testing, as their experiments focused primarily on

Edge Servers with IDSs' performance. The comparisons with related works are also not fully conclusive, as the datasets used in the related works differ. Additionally, the aspect of hardware acceleration was not explored in this paper.

The authors in (Monirah and Ykhlef 2023) propose a **hybrid IDS framework named DeepBlock**, which combines Blockchain and IDSs to detect network attacks. The framework has two main phases, namely, off-chain and on-chain detections. The off-chain detection utilises two techniques, LSTM (as a signature-based IDS) and fuzzy clustering (as an anomaly-based IDS), to learn and generate detection rules from the NetFlow dataset. The on-chain detection phase deploys generated rules from the off-chain phase into the Blockchain. More specifically, these rules are encrypted and stored within smart contracts on a private Blockchain.

The challenges in this novel proposal are storage limitations and potentially high operational costs. Firstly, storing detection rules on smart contracts comes with memory constraints, as Blockchain technology is not designed to process large databases efficiently. Secondly, while reading data from the Blockchain is free, writing detection rules onto the Blockchain incurs increasing costs as the database grows, posing a scalability challenge. Besides, this system has not yet been implemented or evaluated, leaving its practical applicability and effectiveness as an area for future exploration.

(Alkadi et al. 2020) has studied the **combination of Blockchain and IDSs to improve privacy and detect cyberattacks**. They have proposed a deep Blockchain framework to deal with sequential network data; the framework employs the bidirectional LSTM algorithm on the UNSW-NB15 and BoT-IoT (Koroniotis et al. 2019) datasets to detect intruders. Besides, Blockchain and smart contracts are proposed as a privacy-preserving method in trusted execute environments to provide confidentiality. This paper primarily focuses on IDSs to detect attacks, with limited efforts dedicated to evaluating smart contracts. While the paper explores the potential of smart contracts in a secured framework, the practical implementations of these contracts are not thoroughly addressed, particularly in smart contract implementation; there are challenges such as scalability, cost, and storage limitations that must be tested carefully.

(Demertzis et al. 2020) proposes **a deep learning smart contract using an Autoencoder algorithm for anomaly detection in IoT networks**. In this system, smart contracts manage and validate messages by forwarding them to a dedicated cloud service, where message features are extracted as inputs to an

Autoencoder model. If the model detects no attack, the message is forwarded to the target device; otherwise, the message is dropped. The paper experiments on a small dataset, as the computational cost of processing large datasets on a Blockchain is memory-expensive. Besides, the paper lacks a clear explanation of how smart contracts are designed, implemented, and deployed. Additionally, the authors focus on evaluating the detection accuracy of the Autoencoder model without addressing other aspects of the smart contracts, such as the deployment processes of smart contracts, performance metrics, gas costs, and network latency.

5.2.2 Smart Contract in IoT Access Control

Research shows that applying smart contracts to IoT networks has huge potential by providing decentralization, identity, reliability and security in IoT (J. Ali et al. 2020; Reyna et al. 2018; Fotiou and Polyzos 2018).

Smart contracts show advantages in IoT authentication and access control. The researchers in (Cui et al. 2020) propose a Blockchain-based multiWSN authentication scheme for IoT, in which a hybrid Blockchain model has both a local and a public chain. The ordinary node identity authentication is done by the local Blockchain, while cluster head node authentication is done by the public chain. The work in (Algarni et al. 2021) implements a multi-agent system to provide lightweight, decentralized IoT access control security. In (Ren et al. 2021), a decentralized open-trusted access control mechanism based on Blockchain and attribute-based encryption provide effective access control for SDN-IoT applications. The literature (Abubakar et al. 2022) uses smart contracts on the Ethereum network as an additional method in the Two-Factor Authentication (2FA) process. The authors in (Dwivedi, Amin, and Vollala 2023) propose a Blockchain-based framework for data authentication and storage in an IoT environment. The design includes three layers: IoT devices, fog nodes, and IPFS. Smart contracts are deployed at the fog nodes to authenticate and transfer data to the storage located at the IPFS layer.

Besides, smart contracts enhance security and privacy in IoT. In (P. Kumar, R. Kumar, et al. 2021), the authors propose a privacy-preserving and secure framework (PPSF) for smart cities, in which a Blockchain module was used to transmit IoT data securely, and the PCA (Principal Component Analysis) technique was used to transform raw IoT data using a two-level privacy scheme. The paper (Wickström, Westerlund, and Pulkkis 2021) proposes an authentication

model for IoT devices and the distribution of tasks, achieved through two types of smart contracts: manager and device contracts. A protection mechanism is introduced in (H. Shah et al. 2023) with a two-layer security approach for an IoT environment. The first layer filters legitimate users by using ML techniques to filter network packets sent from users. Legitimate users are granted permission to deploy smart contracts. These contracts are fed into ML models, which represent the second layer of security.

5.3 Smart Contracts in IoT Security: Design, Implementation, and Evaluation

Cryptocurrency has gained significant attention thanks to spot Bitcoin's ETF (Exchange-Traded Fund) approval event (Gary Gensler 2024), rendering it accessible worldwide. The technology behind cryptocurrency is called Blockchain, which is a form of DLT (Distributed ledger technology). DLT is a decentralised digital ledger that securely and transparently stores data across multiple nodes. Furthermore, a smart contract is self-executing byte code that runs on Blockchain networks. Smart contracts are triggered by incoming transactions and processed without a central entity. This work utilizes smart contracts to secure IoT network authentication.

Cybersecurity in IoT networks varies according to different perspectives. For instance, edge devices have limited computing resources and are susceptible to malware, side-channel, authentication, and authorisation attacks. Gateway devices, positioned between edge devices and the cloud, are critical points in an IoT network. These devices either collect data or manage other edge devices. Due to their central role, they face higher-level attacks such as network, data and privacy attacks. Building a secured IoT network requires a comprehensive approach across different layers of the architecture.

This chapter proposes a secure and scalable IoT network architecture leveraging Blockchain technology. Specifically, the smart contract was designed and implemented on the Blockchain to enhance security and scalability in IoT networks. The chapter's contributions can be summarised as follows:

- The chapter utilises distributed ledger technology, with smart contracts employed for IoT authentication and authorisation. The smart contract enables two levels of authorisation. The first level allows the contract

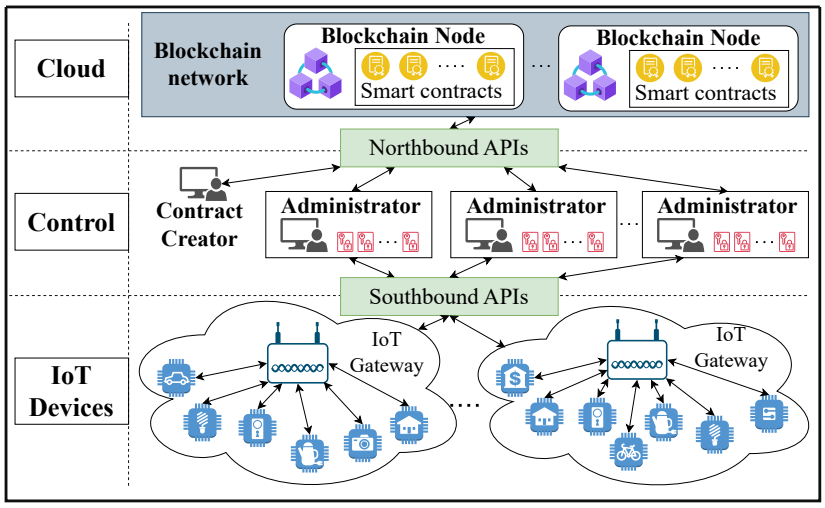


Figure 5.6: The proposed Blockchain-based IoT network architecture with three layers: *IoT Devices*, *Control*, and *Cloud*.

owner to grant permissions to administrators; the second level enables administrators to manage gateway and edge devices in the IoT network.

- The proposed approach is evaluated to demonstrate its scalability and security. The data within the system are managed by the smart contract on the Blockchain, which is transparent and immutable.

5.3.1 Methodology

Figure 5.6 illustrates a high-level view of the proposed architecture[2], comprising three layers, namely *IoT Devices*, *Control*, and *Cloud*.

The *IoT Devices* layer represents the traditional IoT network, where IoT devices are managed in separate groups. Each group has one or more admin devices, called IoT Gateways, to gather data from the edge devices inside the local group. These gateways send the collected data to administrators at the *Control* layer through *Southbound APIs* which facilitates communication between the *Control* and *IoT Devices* layers.

The *Control* layer comprises administrators operating the IoT network. The proposed approach includes two types of administrators with different priority levels. A *Contract Creator* has the highest authority level and is responsible for deploying smart contracts on the Blockchain, and so has full control over the

[2]The figure represents the final state of the work and have evolved from the research work and the ongoing iteration.

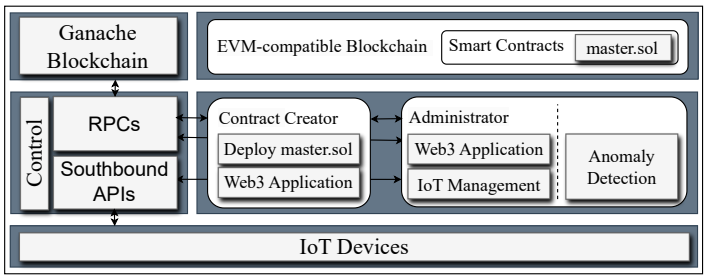


Figure 5.7: The first prototype architecture block diagram.

deployed smart contracts, including managing administrators (adding, removing and editing). In addition to the *Contract Creator*, standard administrators connect with IoT Gateways within the *IoT Devices* layer. These administrators can interact with the Blockchain through *Northbound APIs*.

The *Cloud* layer includes a public Blockchain network that supports smart contracts technology. This technology allows the *Contract Creator* in the *Control* layer to deploy smart contracts on the Blockchain.

5.3.2 Implementation

This section represents the first implemented version of the system from the methodology described in the previous section. Figure 5.7 shows the first prototype architecture, which focuses on smart contracts for IoT security and management. The figure shows the data flow using a smart contract, which enables two levels of authorisation. The first level allows the contract owner to grant permissions to administrators; the second level enables administrators to manage gateway and edge devices in the IoT network. The implementation of the smart contract and processing engines are described as follows:

The *Ganache Blockchain* facilitates deploying smart contracts in the *Cloud* layer. In this version, the *IoT Devices* layer is simulated using software-generated data. The *Control* layer can interact with the *Ganache Blockchain* and *IoT Devices* layers through RPCs (Remote Procedure Calls) and *Southbound APIs*, respectively.

The proposed approach uses two types of administrators in the *Control* layer, namely a *Contract Creator* and standard *Administrator*s, which function as follows:

- The *Contract Creator* block has two modules, namely *Deploy master.sol* and *Web3 Application*, which are used for smart contracts deployment and interaction purposes. Figure 5.8 shows the smart contract's UML class di-

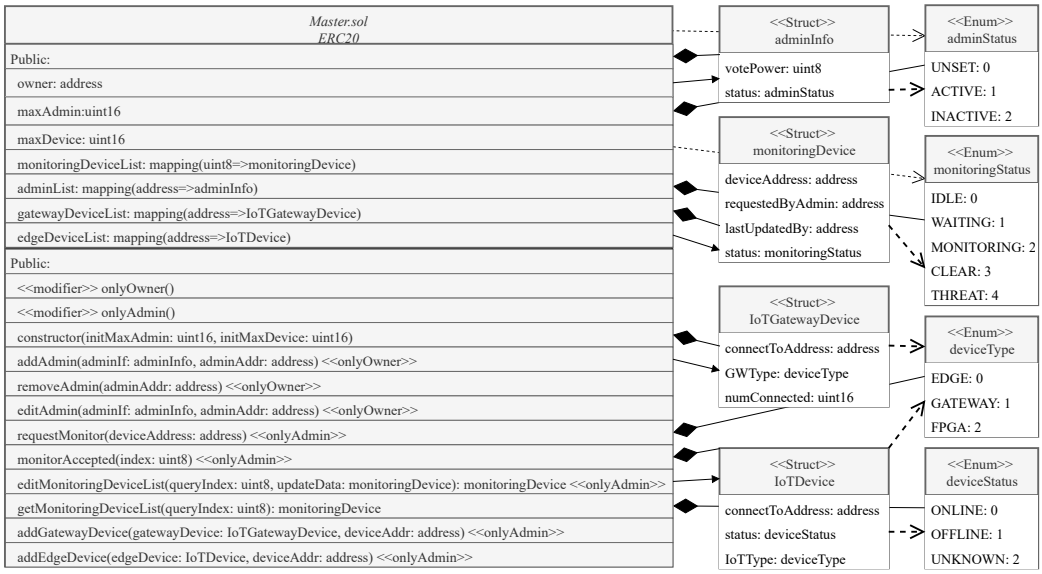


Figure 5.8: The smart contract UML class diagram.

agram (*master.sol*). The smart contract uses *onlyOwner* and *onlyAdmin* modifiers, which represent the *Contract Creator* and the *Administrator* blocks, respectively. While the former modifier supervises and grants permissions to administrators, the latter modifier manages IoT devices. These two modifiers represent two authorisation levels proposed in the *master.sol* smart contract.

- The *Administrator* block contains *Web3 Application*, *IoT management* and *Anomaly Detection* modules. The *Web3 Application* module interacts (including read and write operations) with the smart contract on the Blockchain. The *IoT management* module transmits and receives data from the Blockchain through the *Web3 Application* module. This module is typically employed to develop IoT management applications, such as generating network topology and conducting historical queries. In addition, the *Anomaly Detection* module works closely with the *IoT devices* layer to verify a device before adding it to the Blockchain.

Interacting with smart contracts, using *Web3 Application* transactions, enables developers to build a wide range of applications. Here, three such applications are highlighted, namely (i) adding a new admin, (ii) adding a new IoT device, and (iii) creating a network topology.

1. Before joining the system, each standard administrator must be approved by the *Contract Creator*. Figure 5.9 depicts the process of adding a new

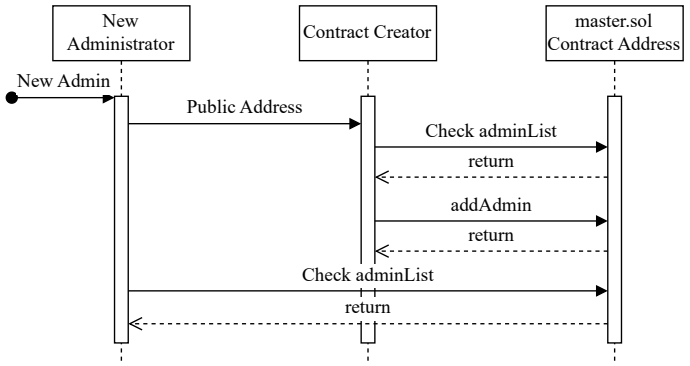


Figure 5.9: Adding a new admin sequence diagram.

admin. The *Administrator* must own a private key associated with its public EVM address. The *Administrator* transmits the public address to the *Contract Creator* as a request to be added to the administrator list; subsequently, the *Contract Creator* queries the smart contract for the current administrator list, to perform policy verification, before incorporating the new address. The *Administrator* can read the smart contract data afterwards to ascertain whether the request was approved or rejected.

2. Updating the IoT device list on the Blockchain must be performed by an approved administrator. Figure 5.10 shows the sequence diagram for adding a new device to the Blockchain. The process commences with an approved administrator sending a device registration request to the smart contract. The new device is added to an under-monitoring list on the Blockchain. The other administrators in the global IoT network can query the device monitoring list to get the target devices; then, the monitored results can be updated to the Blockchain. The new device can be added after the administrator examines the monitored results from the global IoT network.

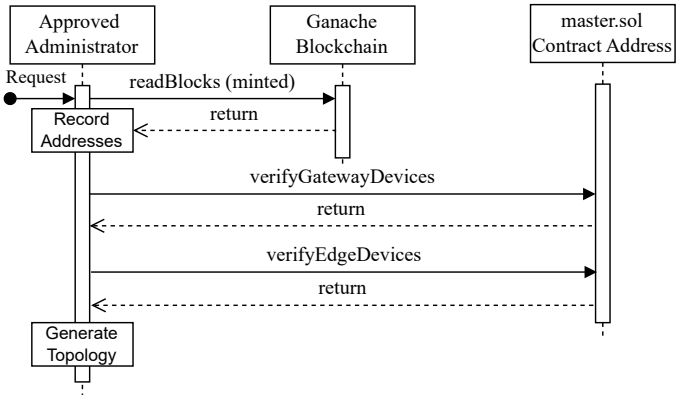


Figure 5.11: The sequence diagram of device verification process.

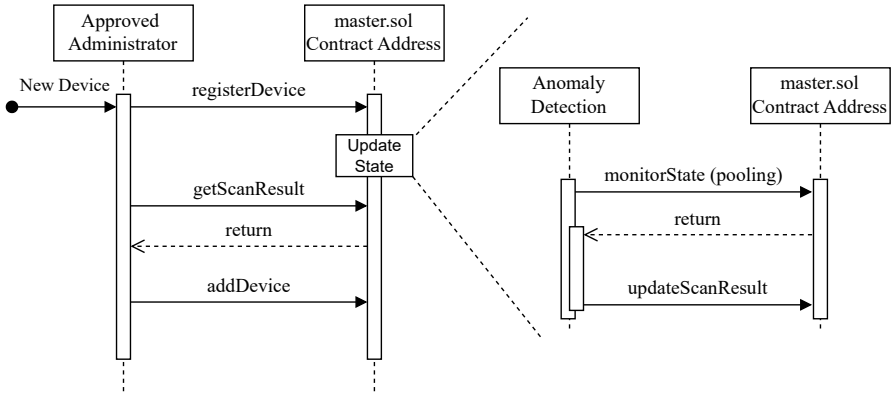


Figure 5.10: Adding a new device sequence diagram.

3. Generating the network topology is one of the essential applications for administrators in the system. The *Administrator* can read and verify the information directly from the Blockchain without the need for central authorisation. Figure 5.11 illustrates the pre-processing steps for gathering and verifying IoT devices' information from the Blockchain. The device addresses are first collected from the blocks mined on the Blockchain. These addresses must be verified by the smart contract before the administrator can generate the topology. The address verification process also includes getting the newest device state and connections.

The *Anomaly Detection* module incorporates lightweight neural network models pre-trained on public IoT datasets. This module, which uses a supervised learning approach, is adapted from the previous work.

5.3.3 Evaluation

The implemented prototype is simulated and tested on a laptop with Intel(R) Core(TM) i7 – 9750H, 2.8GHz x 6 cores, and 16GB of RAM on Windows 10 Pro. Figure 5.12 shows the simulation environment.

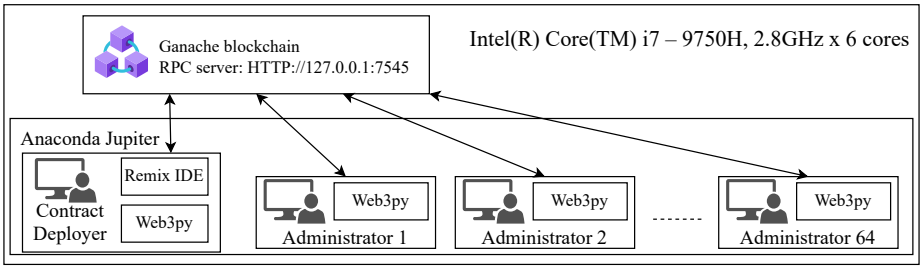


Figure 5.12: The simulation environment with ganache Blockchain running on localhost.

The Ganache Blockchain is configured with version 2.7.1. The hostname, port number, and network ID are 127.0.0.1, 7545, and 5777, respectively. The contract deployer and administrators are running on Jupiter Notebooks under the Anaconda toolbox.

The smart contract is compiled and deployed using Remix IDE (Remix 2024). The smart contract is initialised with the maximum number of administrators and devices as 63 and 65,535, respectively. The contract deployment transaction used 2,891,713 gas on ganache software.

Next, the total time needed to generate the network topology is measured. This value is contributed by two processes: reading minted blocks to gather devices' addresses and verifying these addresses on the Blockchain to get the latest data. The test created scenarios with different numbers of administrators and IoT devices. In this experiment, the number of administrators in the system is increased by one at a time (from 1 to 10). Each administrator manages x gateway devices while a gateway device has y edge devices connected to it. These numbers, x and y, are generated randomly in the range [1:10] and [1:100], respectively. Table 5.3 shows the experimental results in detail. The first column is the number of administrators; the second and third columns are the total number of gateways and edge devices. The last two columns are block query and verification time values in seconds.

Table 5.3: The time for querying block history and verifying devices on the Blockchain.

Administrators	Gateways	Edge devices	Block query (s)	Verification (s)
1	8	52	2.57	1.46
2	14	90	2.68	1.45
3	21	184	3.49	2.55
4	24	257	3.88	3.63
5	34	341	4.83	5.03
6	35	399	5.53	5.76
7	38	482	6.59	6.12
8	47	554	6.82	7.23
9	55	633	7.71	8.10
10	62	730	8.23	9.53

The smart contract was deployed at block 353, and the test was finished at block 1,155. The gas used in one transaction for adding an administrator, a gateway device and an edge device are 51,634, 48,678 and 48,647, respectively. Initially, there was one administrator and 60 devices (8 gateway and 52 edge devices). The corresponding time for block query and verification processes were 2.57 and 1.46 seconds. These values increased gradually as more administrators and devices were added. When 10 administrators and 792 devices were added, the block query time was 8.23 seconds, and the verification time was 9.53 seconds.

Figure 5.13 illustrates the generated topology of devices in the first scenario, in which the administrator manages eight gateway and 52 edge devices. Each node in the topology is marked with an integer number, which is mapped with an EVM address.

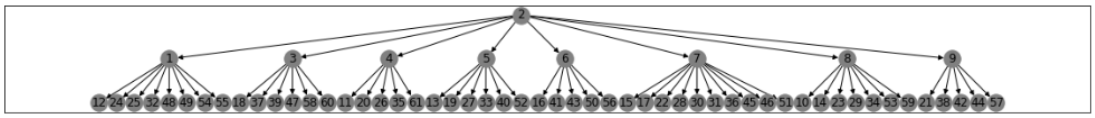


Figure 5.13: The generated topology in the first scenario with the data from the Blockchain.

The experiment tests the smart contract with more edge devices and records the transaction time and the total gas cost. Table 5.4 summarises the results with the number of edge devices ranging from 1,024 to 65,535.

Table 5.4: The total transaction time and gas cost when adding more devices.

Edge devices	Transaction time (s)	Total gas used
1,024	28.79	31,364,892
2,048	57.82	81,167,692
4,096	115.83	180,812,912
8,192	231.78	380,103,076
16,384	476.40	778,759,304
32,768	982.20	1,576,115,104
65,535	2,133.65	3,170,828,660

The total time required for the smart contract to add new devices into the system increased steadily when the number of devices doubled. These numbers are 28.79 seconds and 2,133.65 seconds when adding 1,024 and 65,535 devices, respectively. The total gas used in the test is 3,170,828,660; with a gas price of 1.0, the total Ethereum consumed on the ganache Blockchain is 3.22 ETH.

5.3.4 Discussion

This section explains the design choices and then discusses the proposed system's advantages and disadvantages. Firstly, the data generated from IoT devices is extremely voluminous over time; thus, storing these data on the Blockchain is costly. The smart contract is proposed to store management data on-chain, specifically only information about administrators, devices, and their connections, to minimise the operation cost. Furthermore, each administrator and IoT device in the system must have an EVM key pair to participate in the system. This enhances security by preventing attackers from impersonating devices or administrators. Moreover, this allows smart contract variables to be designed using the mapping type rather than arrays, thereby reducing internal data processing and querying fees.

Secondly, while the contract owner in the proposed system holds the highest level of security, this also introduces a potential vulnerability. This issue can be mitigated by applying the multi-signature technique, where a group of individuals shares equal permission, and each task's execution in the smart contract must be approved by the majority. Additionally, Blockchain's transaction history is a valuable and powerful resource. Reading a public Blockchain history

does not consume gas. The proposed approach is implemented and tested using Web3 applications to generate network topology by querying Blockchain history with zero gas cost.

Finally, the Anomaly Detection module in this version is still in the simulation phase. This module focuses on detecting attacks using lightweight machine learning models (Artificial Neural Networks and Convolutional Neural Networks). Specifically, these models will be implemented using hardware accelerators. IoT security and management are enhanced through smart contracts on the Blockchain. Moreover, computational resources are shared as network administrators can contribute to attack detection from the edge to the Blockchain on the cloud.

5.4 Summary

This chapter studied distributed-ledger technology, which is the technology behind Blockchain and smart contracts. Its widespread adoption has made this technology a cornerstone for many decentralised applications, particularly those related to digital currency. The chapter specifically focused on its characteristics, including immutability, transparency, and accessibility, to develop a scalable security system in IoT networks.

The chapter has conducted a systematic study of relevant work that leverages smart contracts for IoT security. Additionally, to test the proposed approach, Blockchain platforms that support smart contracts have been explored, along with tools and programming languages used for the development of Blockchain and smart contracts.

a scalable security Blockchain-based architecture for IoT networks has been proposed. The approach utilises Blockchain and smart contracts for IoT authentication and authorisation. The implemented design promises a consistent transaction gas cost for system scalability. The Blockchain provides administrators with security and specific control over devices in the network. The experimental results showed that the implemented prototype manages the administrators and devices according to defined terms in the smart contract, which only authorised devices can access and change the system.

In the next chapter, high-speed anomaly detection using lightweight machine-learning models on hardware accelerators will be discussed. Furthermore,

adding the multi-signature function and proxy feature can improve security and scalability for smart contracts. The next chapter will combine the proposed approaches, including security for IoT networks using smart contracts on the cloud and anomaly-based IDSs at the edge.

References

Abubakar, Mwrwan et al. (2022). "A lightweight and user-centric two-factor authentication mechanism for IoT based on blockchain and smart contract". In: *2022 2nd International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*. IEEE, pp. 91–96.

Algarni, Sultan et al. (2021). "Blockchain-based secured access control in an IoT system". In: *Applied Sciences* 11.4, p. 1772.

Ali, Jawad et al. (2020). "Towards secure IoT communication with smart contracts in a blockchain infrastructure". In: *arXiv preprint arXiv:2001.01837*.

Alkadi, Osama et al. (2020). "A deep blockchain framework-enabled collaborative intrusion detection for protecting IoT and cloud networks". In: *IEEE Internet of Things Journal* 8.12, pp. 9463–9472.

Consortium, ns-3 (2024). *ns-3 Network Simulator*. `https://www.nsnam.org/`. Accessed: Dec 16, 2024.

Crypto Atlas (2024). *Aptos VS. Solana: Which One is Better?* Accessed: Dec 16, 2024. URL: `https://cryptoatlas.net/63-aptos-vs-solana.html`.

Cui, Zhihua et al. (2020). "A hybrid blockchain-based identity authentication scheme for multi-WSN". In: *IEEE Transactions on Services Computing* 13.2, pp. 241–251.

Demertzis, Konstantinos et al. (2020). "Anomaly detection via blockchained deep learning smart contracts in industry 4.0". In: *Neural Computing and Applications* 32.23, pp. 17361–17378.

Dwivedi, Sanjeev Kumar, Ruhul Amin, and Satyanarayana Vollala (2023). "Smart contract and IPFS-based trustworthy secure data storage and device authentication scheme in fog computing environment". In: *Peer-to-Peer Networking and Applications* 16.1, pp. 1–21.

Eskandari, Mojtaba et al. (2020). "Passban IDS: An intelligent anomaly-based intrusion detection system for IoT edge devices". In: *IEEE Internet of Things Journal* 7.8, pp. 6882–6897.

Ethereum (2024). *Introduction to smart contracts*. `https://ethereum.org/en/smart-contracts/` visited on Dec 16, 2024.

Ethereum Foundation (2022). *Ethereum's Transition to Proof of Stake (The Merge)*. `https://ethereum.org/en/roadmap/merge/`. Accessed: Dec 16, 2024. URL: `https://ethereum.org/en/roadmap/merge/`.

Fotiou, Nikos and George C Polyzos (2018). "Smart contracts for the internet of things: Opportunities and challenges". In: *2018 European conference on networks and communications (EuCNC)*. IEEE, pp. 256–260.

Gary Gensler (2024). *Statement on the Approval of Spot Bitcoin Exchange-Traded Products*. https://www.sec.gov/news/statement/gensler-statement-spot-bitcoin-011023 visited on Dec 16, 2024.

Github (2024). *The Move Language on Aptos*. Accessed: Dec 16, 2024. URL: https://github.com/move-language/move-on-aptos.

Hyperledger Foundation (2024a). *Hyperledger Fabric*. Accessed: Dec 16, 2024. Linux Foundation. URL: https://www.hyperledger.org/use/fabric.

– (2024b). *Writing Your First Chaincode*. Accessed: Dec 16, 2024. URL: https://hyperledger-fabric.readthedocs.io/en/latest/chaincode4ade.html.

Jha, Shyamli (2023). *The Complete Guide for Types of Blockchain*. https://www.simplilearn.com/tutorials/blockchain-tutorial/types-of-blockchain visited on Dec 16, 2024.

Koroniotis, Nickolaos et al. (2019). "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset". In: *Future Generation Computer Systems* 100, pp. 779–796.

Kumar, Prabhat, Randhir Kumar, et al. (2021). "PPSF: A privacy-preserving and secure framework using blockchain-based machine-learning for IoT-driven smart cities". In: *IEEE Transactions on Network Science and Engineering* 8.3, pp. 2326–2341.

Linux Foundation (2016). *Hyperledger Project*. https://www.hyperledger.org/. Accessed: Dec 16, 2024. Linux Foundation. URL: https://www.hyperledger.org/.

Monirah, AI-Ajlan and Mourad Ykhlef (2023). "DeepBlock: a Collaborative Intrusion Detection Framework Based on Blockchain and Deep Learning". In: *2023 Fifth International Conference on Blockchain Computing and Applications (BCCA)*. IEEE, pp. 180–185.

Nakamoto, Satoshi (2009). "Bitcoin open source implementation of P2P currency". In: *February* 11.2009, P2P.

NEVIL, SCOTT (2024). *Distributed Ledger Technology (DLT): Definition and How It Works*. https://www.investopedia.com/terms/d/distributed-ledger-technology-dlt.asp visited on Dec 16, 2024.

Nguyen, Xuan-Ha et al. (2022). "Realguard: A lightweight network intrusion detection system for IoT gateways". In: *Sensors* 22.2, p. 432.

Remix (2024). *REMIX PROJECT*. https://remix-project.org/ visited on Dec 16, 2024.

Ren, Wei et al. (2021). "SILedger: A blockchain and ABE-based access control for applications in SDN-IoT networks". In: *IEEE Transactions on Network and Service Management* 18.4, pp. 4406–4419.

Reyna, Ana et al. (2018). "On blockchain and its integration with IoT. Challenges and opportunities". In: *Future generation computer systems* 88, pp. 173–190.

Shah, Harshit et al. (2023). "Deep learning-based malicious smart contract and intrusion detection system for IoT environment". In: *Mathematics* 11.2, p. 418.

Sharadqh, Ahmed AM et al. (2023). "Hybrid chain: Blockchain enabled framework for bi-level intrusion detection and graph-based mitigation for security provisioning in edge assisted IoT environment". In: *IEEE Access* 11, pp. 27433–27449.

Solana Foundation (2024). *Anchor - Solana Sealevel Framework*. Accessed: Dec 16, 2024. URL: https://github.com/coral-xyz/anchor#readme.

Solidity Team (2024). *Solidity - A statically-typed curly-braces programming language designed for developing smart contracts that run on Ethereum*. https://soliditylang.org/ visited on Dec 16, 2024.

TRUFFLE SUITE (2024). *Ganache - ONE CLICK BLOCKCHAIN*. https://archive.trufflesuite.com/ganache/ visited on Dec 16, 2024.

Wickström, John, Magnus Westerlund, and Göran Pulkkis (2021). "Smart Contract based Distributed IoT Security: A Protocol for Autonomous Device Management". In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 776–781. DOI: 10.1109/CCGrid51090.2021.00094.

Chapter 6

Framework Design and Implementation

This chapter presents the publication titled "A Scalable Security Approach in IoT Networks: Smart Contracts and Anomaly-based IDS for Gateways using Hardware Accelerators," which combines the continuous developments from the previous chapters, including the implementation and evaluation of the proposed approach. The chapter's contributions can be summarised as follows:

The chapter proposes a network architecture for IoT networks comprising three layers, namely cloud, control, and IoT devices. The proposed architecture enhances system security and scalability by leveraging smart contracts technology and ML on hardware accelerators. Secured functions are designed and integrated into each layer.

The smart contract is designed and implemented on the cloud layer to manage access control and verify IoT devices. IoT management data is stored on smart contracts in a decentralised network, in which device authentication is executed automatically and recorded on the Blockchain. The proposed approach describes an FPGA-based architecture for Gateways, which integrates lightweight ML models to detect attacks at the device level. Furthermore, the feature extraction modules are implemented on FPGA to reduce overhead relative to CPU and GPU schemes.

Finally, the proposed approach is evaluated using various testing scenarios. Smart contracts are deployed on a simulated Ethereum Blockchain network. A setup is built to simulate the smart contract connections and operations. The Gateway is implemented and tested on a NetFPGA-PLUS platform (Alveo U280

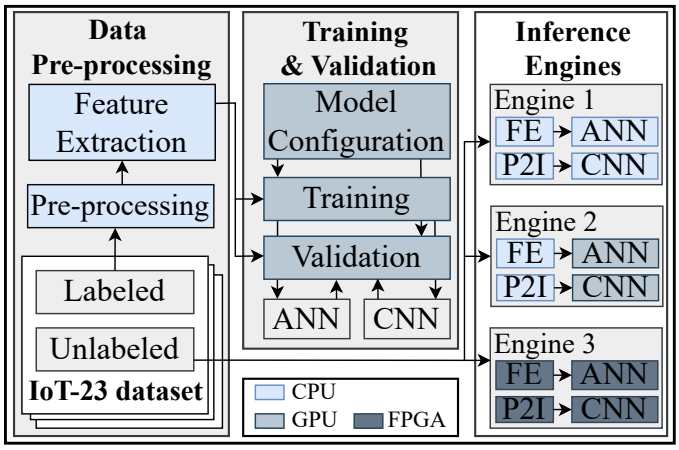


Figure 6.1: The platform-dependent anomaly-based IDS framework on the IoT-23 dataset.

card (AMD 2021)). The system is evaluated on the IoT-23 dataset (Garcia, Parmisano, and Erquiaga 2020b). Hardware resource usage and power consumption are also discussed.

6.1 Anomaly-based IDS on Hardware Platforms

The anomaly-based IDS has been adapted from the previous chapters. The implemented system uses lightweight ML models to detect network attacks based on a packet's header, including IP addresses, TCP/IP ports and protocol. Figure 6.1 illustrates the system's processing flow on the IoT-23 dataset (Garcia, Parmisano, and Erquiaga 2020b).

The proposed anomaly-based IDS is platform-dependent and comprises three stages, namely *Data pre-processing*, *Training & Validation* and *Inference Engines*. The following sections describe the dataset and framework processing flow in detail.

6.1.1 Data Pre-processing

The *Data Pre-processing* stage prepares input features for the next stage on the CPU platform. The feature extraction process is different for ANN and CNN models. While the ANN model uses packet header fields as input, the CNN model requires a converting process in which network packets are converted to fixed-size images.

6.1.2 Training and Validation

The training and validation processes are conducted on the GPU platform. The ANN and CNN models are tested using different configurations to produce lightweight models with high detection rates. In particular, the ANN model is configured with 32 neurons in the hidden layer, each followed by a Rectified Linear Unit (ReLU) activation function. The CNN model architecture includes four convolutional layers, two max pooling and two-platen layers at the end (Ngo, Lightbody, Temko, Pham-Quoc, et al. 2023b).

6.1.3 Inference Engines

The inference stage will be evaluated with three engines using different hardware accelerator combinations. Each engine has a Feature Extraction (FE) and a Packet-to-Image (P2I) module as data pre-processing for the ANN and CNN models, respectively. The engines can be described as follows:

- Engine 1 only uses CPUs without any other hardware accelerator.
- Engine 2 uses a GPU for running the models while the data pre-processing tasks are assigned to CPUs.
- Engine 3 is implemented on an FPGA-based architecture.

The processing performance of these engines will be discussed in Section 6.3.4.

6.2 Proposed Architecture

This section introduces an overview of the proposed architecture and describes the system implementation in detail.

6.2.1 Overview

Figure 6.2 illustrates the proposed architectural overview with three layers, which are *Cloud*, *Control* and *IoT Devices*.

The *Cloud* layer presents the top level in the proposed architecture in which the *Blockchain Nodes* constitute a Blockchain network. The Blockchain network represents a distributed digital ledger that supports smart contract technology. Smart contracts are proposed for managing access controls and security in IoT networks.

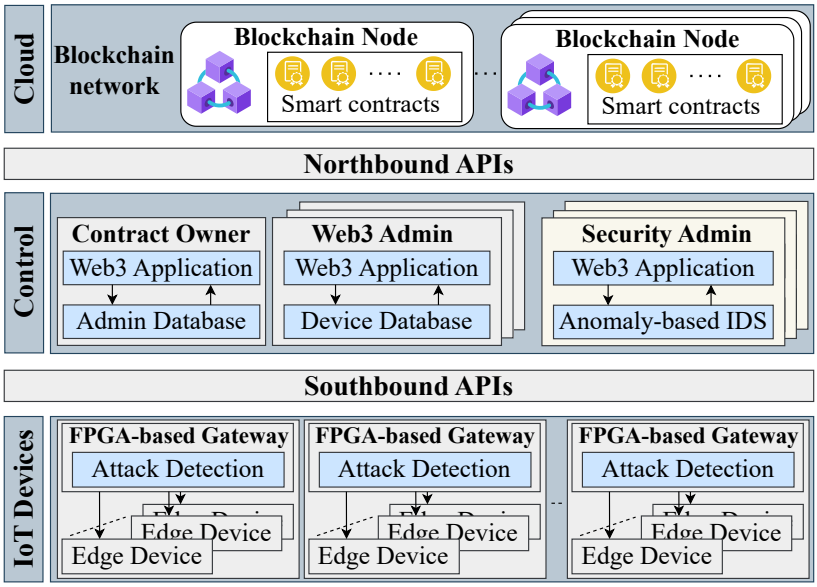


Figure 6.2: The proposed architecture with three layers: *IoT Devices*, *Control* and *Cloud*.

The *Control* layer is the intermediate layer to either interact with the Blockchain network or manage IoT devices through Northbound APIs and Southbound APIs, respectively. This layer comprises different administrators who have different roles in the system. A *Web3 Application* module indicates that the administrator can interact with the Blockchain using Web3 technology. As shown in Figure 6.2, administrators are presented as blocks and can be described as follows

- The *Contract Owner* block is the administrator with the highest authority level to deploy smart contracts onto the Blockchain network and so has full control over the deployed smart contracts. The contract owner owns a local *Admin Database* to manage administrators through the *Web3 Application* module.
- The *Web3 Admin* blocks focus on organising IoT gateways and edge devices.
- The *Security Admin* blocks are administrators who are equipped with hardware accelerators for training and validating attack detection models.

The *IoT Devices* layer represents an IoT network where IoT devices are organised in groups. Each group is proposed to have one FPGA-based gateway device to

gather data from the edge devices. The gateway sends the collected data to administrators at the *Control* layer through the communicating channel, called *Southbound APIs*.

6.2.2 Implementation

This section describes the protocols and technologies used to implement the proposed architecture. This involves the Blockchain and smart contract in the *Cloud* layer, the Web3 application modules in the *Control* layer and the FPGA-based Gateways in the *IoT Devices* layer.

6.2.2.1 Web3 Applications

The *Web3 Application* module is the compulsory module used by each administrator to communicate with the smart contracts on the Blockchain. *Web3 Application* modules corresponding to the three administrator groups in the *Control* layer are discussed as follows

- The *Contract Owner* has a local database to store the administrators' information in the network. Administrators must first send their EVM address to the contract owner, who then decides whether to approve this address as a legitimate administrator in the smart contract. Figure 6.3 depicts the sequence diagram for adding a new administrator process.

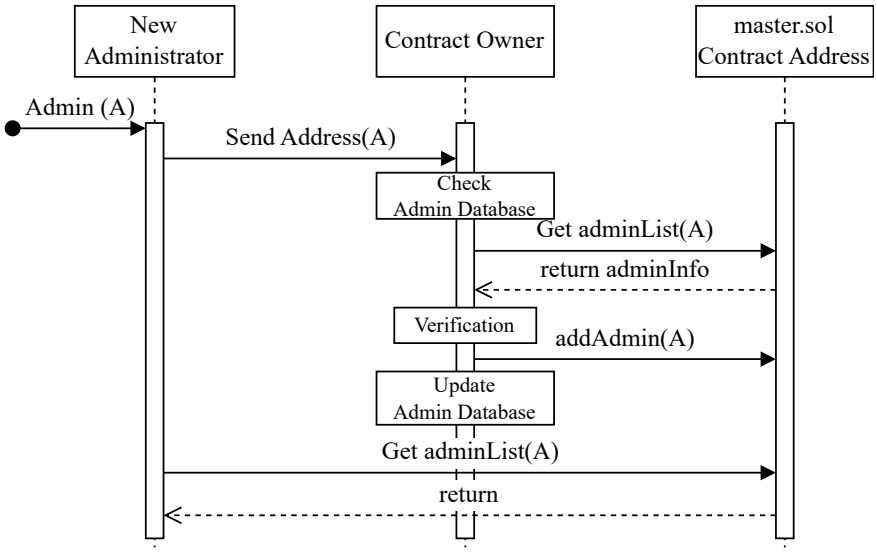


Figure 6.3: The sequence diagram for adding a new administrator (denoted as (A)) process.

Initially, the new administrator sends its own EVM address to the contract owner. Then, the contract owner works with the local database before getting the latest information about the received address on the Blockchain. If the address has not been added, the contract owner can send a transaction to add the new administrator and update the local database afterwards. The new administrator can verify the result by reading the smart contract.

- The *Web3 Admin* also has a device database, which manages the status and EVM addresses of the new IoT devices. Figure 6.4 illustrates the sequences from registering to security validation when adding a new device to the network.

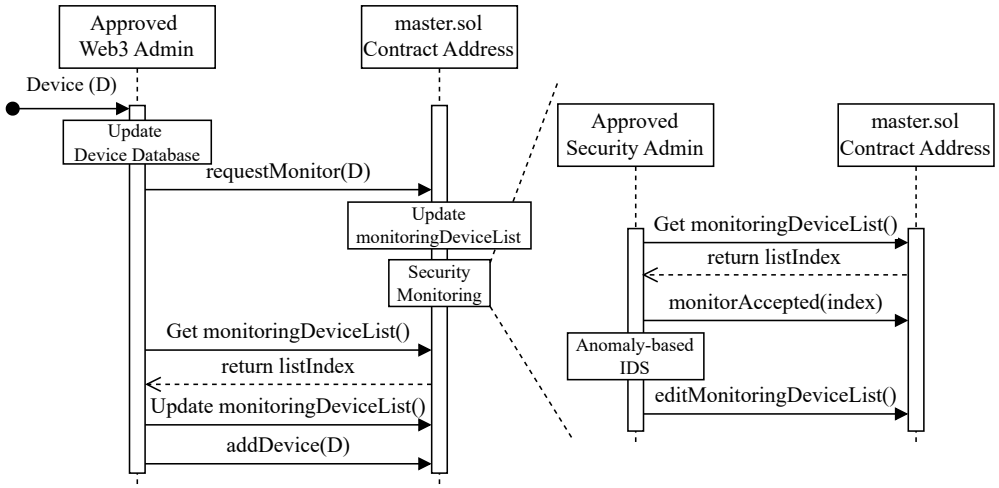


Figure 6.4: The sequence diagram for adding a new device (denoted as D) to the network.

An approved administrator first registers the new device with an EVM address with the smart contract. The smart contract then waits for the security monitoring process, which updates the device status by an approved security admin. This process includes two main steps: Step 1 is registering the device in the queue, and Step 2 is waiting for verification by the security module.

- The *Security Admin* is proposed with an *anomaly-based IDS* module, which is adapted from the previous chapter. These administrators are responsible for training and validating lightweight attack detection models. The pre-trained model will be accelerated on FPGA hardware. Besides, a *Web3 application* module is used to update the smart contract if there are alerts

from Gateways about monitoring devices.

6.2.2.2 Gateways

This work proposes high-speed FPGA hardware as a platform for Gateways to detect network attacks. In this implementation, the proposed FPGA-based architecture is wrapped in the NetFPGA-PLUS platform (NetFPGA 2021). The NetFPGA-PLUS platform includes Xilinx Alveo Data Center Accelerator Cards based upon Xilinx Virtex Ultrascale+ FPGAs. The platform code base operates on the U200, U250 and U280 cards. Figure 6.5 depicts the NetFPGA-PLUS Gateway architecture that can be described as follows

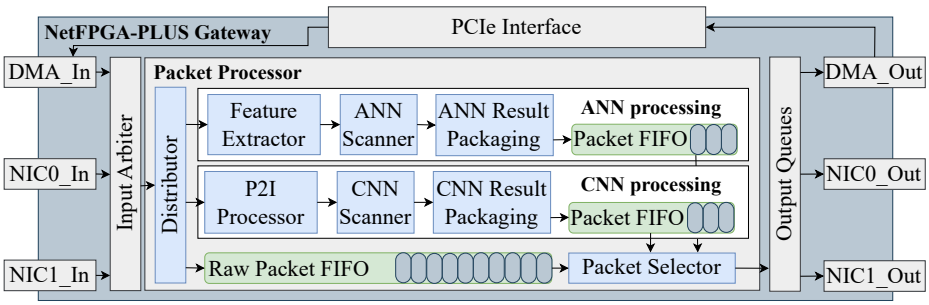


Figure 6.5: The block diagram of the NetFPGA-PLUS Gateway with security integration.

- The *PCIe interface* block represents the communication between a host machine and the NetFPGA-PLUS card through a Peripheral Component Interconnect Express (PCIe) interface; this block simulates virtual network interfaces, which are controlled by the host machine. These virtual interfaces are named DMA_In and DMA_Out, presenting the input/output interfaces to/from the NetFPGA-PLUS card, respectively.
- The *NIC* blocks, including NIC0 and NIC1, are physical network interfaces on the NetFPGA-PLUS card.
- The *Input Arbiter* block operates based on a queue mechanism that uses the round-robin algorithm to select incoming packets from different input interfaces. In other words, this module merges network packets from multiple input sources, including virtual and physical interfaces, to create an output stream of packets.
- The *Packet Processor* block has scanner modules, which accommodate pre-trained lightweight neural network models. This block will be discussed in detail in the next Section.

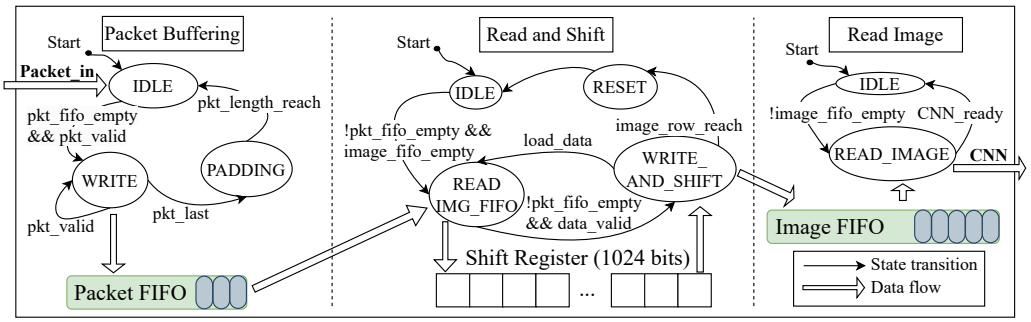


Figure 6.6: The *P2I processor* module state machines and data flow.

- The *Output Queues* block operates inversely to the *Input Arbiter* block described above; this block receives the processed packet stream and distributes it to the corresponding output interfaces.

6.2.2.3 Packet Processor

As shown in Figure 6.5, the *Packet Processor* block is implemented in the NetFPGA-PLUS Gateway between the *Input Arbiter* and *Output Queues* block. This block either forwards network packets or detects network attacks using neural network models. The network packets are processed as follows.

- The *Distributor* module receives the network packets from the *Input Arbiter* block; it then distributes the data simultaneously to three different modules, which are *Feature Extractor (FE)*, *P2I Processor* and *RAW packet FIFO*.
- The *ANN processing* contains *FE*, *ANN Scanner*, *ANN Result Packaging* and *Packet FIFO* modules. Initially, network packets are passed through the FE module to extract header fields (source and destination IP addresses, ports and protocol). These extracted data serve as the input for the *ANN Scanner* module. The scanned results are packaged with the extracted header fields and stored in a *Packet FIFO*.
- *CNN processing* has a similar structure to the *ANN processing*. However, instead of only examining header fields, the CNN model processes the entire network packet, including header and payload. Figure 6.6 illustrates the *P2I processor* module, which converts network packets into images.

  The module is implemented on the FPGA-based architecture using three state machines, namely *Packet Buffering*, *Read and Shift* and *Read Image*, to transform network packets of different lengths into fixed-size images. The

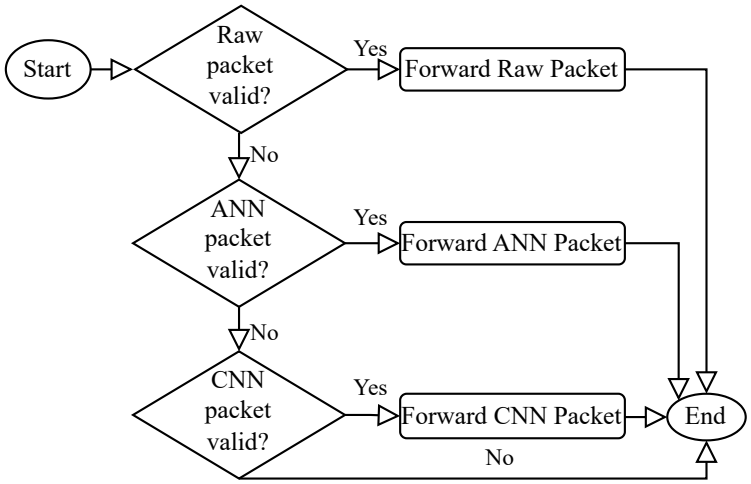


Figure 6.7: The *Packet Selector* module's flow chart expressing packet priority order.

Packet Buffering state machine receives network packets and stores them in a *Packet FIFO*. Given that a network packet length is not fixed, this state machine standardises all network packets to a fixed length (1500 bytes) using the zero padding method. Next, the *Read and Shift* state machine uses a shift register to read the packet frames from the *Packet FIFO* and write this data as image rows to the *Image FIFO*. Finally, the *Read Image* state machine reads the image data from the *Image FIFO* and sends it to the *CNN Scanner* module. The scanned results are packaged with the packet header and stored in a *Packet FIFO*.

- The *Packet Selector* module merges packets from different sources, including original network packets and alarm packets from the *ANN processing* and *CNN processing*. The output is a network packet stream routed to the *Output Queues* block. Figure 6.7 illustrates the packet's priority processing order. Integrating additional security functions is considered to add overhead to the system. Therefore, forwarding raw packets has the highest priority in this implemented version, followed by the result packets from the ANN and CNN scanners.

6.2.3 Design Verification

Design Verification (DV) is an essential process in FPGA design and plays a crucial role in the semiconductor industry before producing the chip. The DV

process has been standardised through different protocols, depending on each semiconductor company. From an academic perspective, design verification tends to be simpler and is project-focused rather than being developed into an industry-wide protocol. This section describes the DV processes for validating FPGA implementations and presents the methodology used to measure the reliability of ANN and CNN models on FPGA hardware.

For small FPGA designs, conventional testing using a testbench with multiple test cases is typically sufficient to evaluate the design correctness, as such designs do not encounter issues related to the FPGA place-and-route process, given the available hardware resources. However, more complex designs that involve the integration of multiple modules introduce additional challenges that can impact the accuracy of the entire system. The proposed system faces similar challenges, as the modules are interconnected in a unified processing flow, where each module seamlessly utilises the output of the preceding module. Therefore, ensuring computational accuracy requires a systematic and carefully prepared evaluation process.

Below, the DV is divided into two subsections corresponding to the two ML models implemented on FPGA: ANN and CNN. The rationale for this division lies in the differences in their implementation methods. The ANN model is optimised by the pipelining method using Verilog. In contrast, the CNN model is implemented using the HLS method, which involves the software-based generation of Register Transfer Level (RTL) code. The DV process is validated using the *reliability metric*, which indicates the similarity in results of an FPGA compared to CPUs.

6.2.3.1 ANN model

Figure 6.8 describes two stages to measure the reliability metric of the ANN model on FPGA. This process begins by utilising the pre-trained ANN model on CPUs, represented by checkpoint files. Then, the model's parameters are generated in NumPy format (Weights.npy). This file encapsulates the ANN model and will be used in the following two stages as follows:

- Stage one, also known as the simulation stage, involves testing the design through simulation by using a small dataset that allows manual observation of each computational step. The input data consists of network packets (test_small.pcap), which are fed into the *Inference Engine* on CPUs, where the layers' outputs are monitored. Simultaneously, on the FPGA

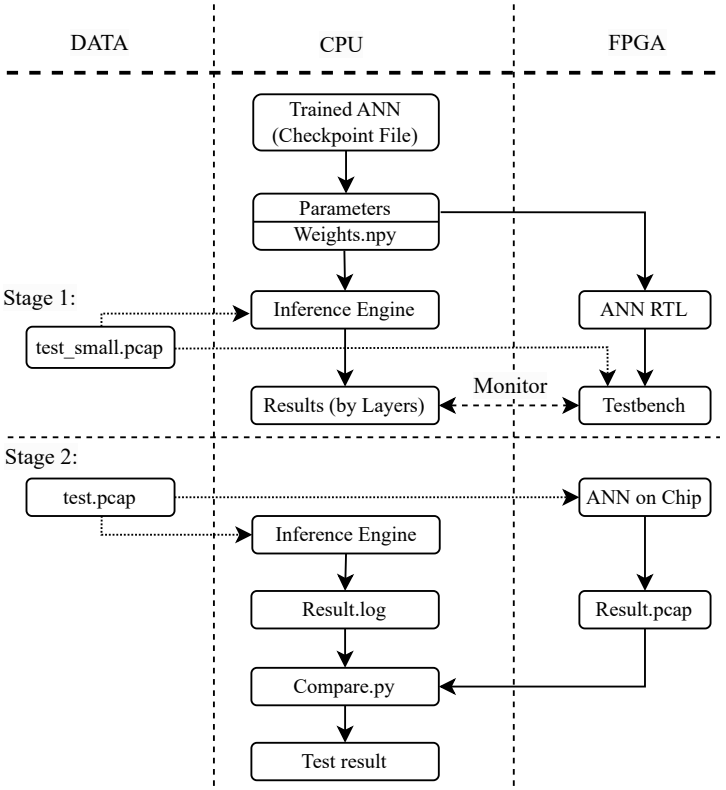


Figure 6.8: Two stages of the design verification process on an ANN model.

design, the same parameters are used for the ANN model (ANN RTL); the network packets are fed into a Testbench. The computational results are monitored and compared between the two platforms, using the CPU's output as the reference.

- Stage two involves testing the actual computational results on the FPGA hardware. The first stage confirms the simulation results for the FPGA on the small dataset; then, a larger portion of the dataset is used to measure the reliability metric. A similar computation process is performed on the CPUs to collect reference results (Result.log). At this point, the ANN model is deployed on an FPGA; the results are captured and stored as network packets (Result.pcap). Then, a script (compare.py) is used to compare the results from both platforms and report the reliability metric.

6.2.3.2 CNN model

Similarly, Figure 6.9 illustrates the process of measuring the reliability metric of the CNN model on FPGA.

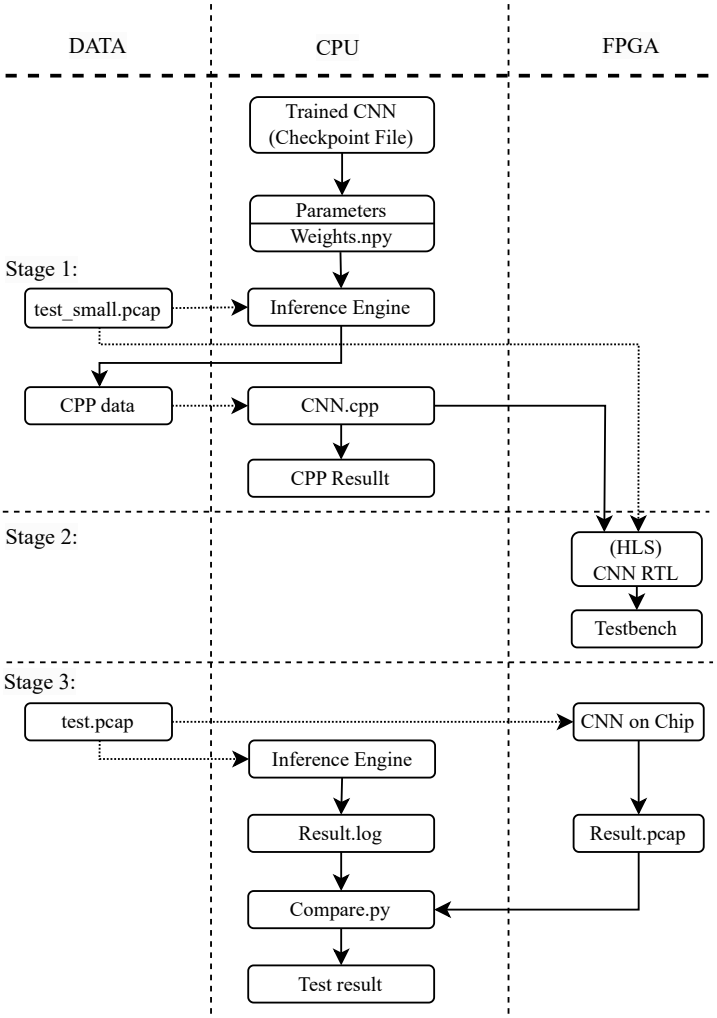


Figure 6.9: Three stages of the design verification process on a CNN model.

The CNN implementation on FPGA used the HLS methodology, which generates RTL from the *CNN.cpp*. So, an extra stage is added to validate the CNN.cpp design. In this stage, the *Inference Engine* generates *CPP data* from the small dataset for validating the CNN model. The result from this stage can be monitored manually. Stages two and three are similar to stages one and two in the previous section, respectively.

6.3 Experiments

This section evaluates the implemented system by layers. Firstly, the experimental setup is introduced, simulating the system in a practical environment. Next, the standalone experiments on the FPGA-based Gateway are performed. Finally, the smart contract on the cloud and the full system are tested.

6.3.1 Setup

Figure 6.10 depicts the setup for testing the implemented system. The setup is a direct result of the research and implementation conducted in this thesis, reflecting the methodologies and design choices developed throughout this study. The setup includes four main host machines, which are described below.

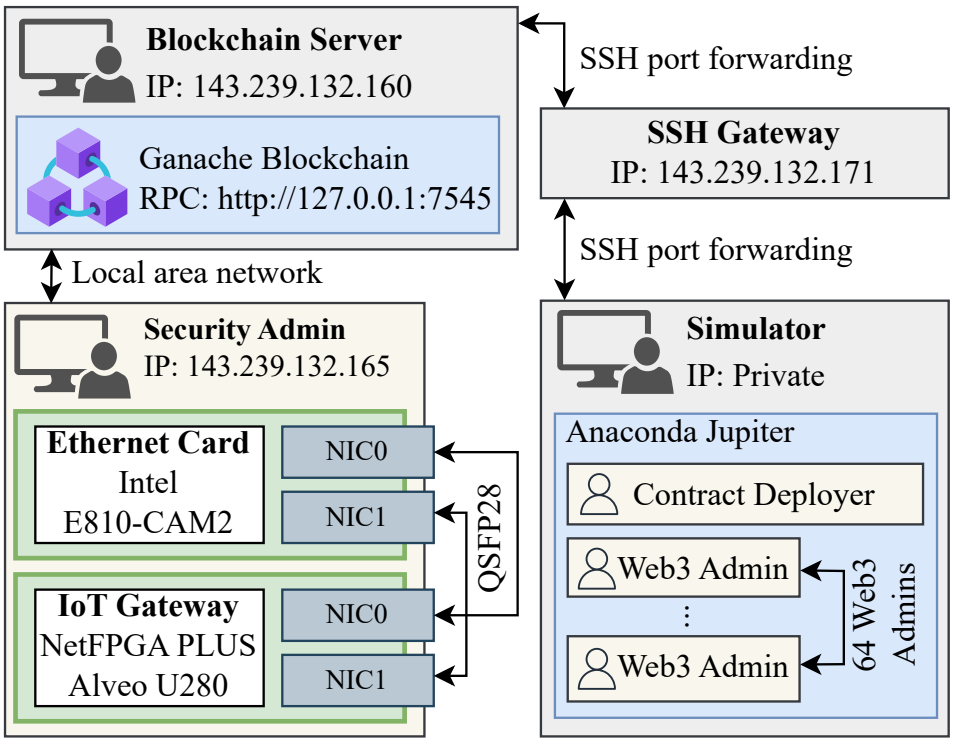


Figure 6.10: The setup with the full system in the physical network.

- The *Blockchain Server* runs on the Dell Precision 7920 Tower with 512GB of memory and Intel(R) Xeon(R) Platinum 8276 CPU @ 2.2GHz x 112 on Ubuntu 22.04.2 LTS operating system. The server has Ganache Blockchain configured with version 2.7.1. The hostname, port number and network

ID are 127.0.0.1, 7545 and 5777, respectively. In addition, the host machine is equipped with dual NVIDIA Quadro GV100 GPUs.

- The *SSH Gateway* (Intel(R) Core(TM) i7 – 10700, 2.90GHz, 32GB of RAM on Rocky Linux 8.10) acts as the access gateway for devices on the internet to the Blockchain server.
- The *Security Admin* (Intel) contains an Ethernet (Intel E810-CAM2) and a NetFPGA-PLUS card (Alveo U280); each has two QSFP28 transceivers. The Ethernet card is directly connected to the NetFPGA-PLUS card using QSFP28 cables for monitoring network packets at high speed. The NetFPGA-PLUS card is programmed as an IoT Gateway and is integrated into the host machine through a PCIe interface. The administrator can interact with the Blockchain server through a LAN (Local Area Network) or an SSH Gateway.
- The *Simulator* (Intel(R) Core(TM) i7 – 9750H, 2.8GHz x 6 cores, and 16GB of RAM on Windows 10 Pro) demonstrates the contract creator and Web3 administrators using Jupiter Notebooks under the Anaconda toolbox.

In this setup, the *Blockchain Server* simulates the cloud layer that runs the Blockchain and smart contracts. This host machine also has GPUs for training, verification and inference of ML models. The *SSH Gateway* works as a tunnel, providing access to the Blockchain so that the *Simulator*, which has a private IP address, can interact with the deployed smart contracts on the *Blockchain Server*.

6.3.2 FPGA-based Gateway

6.3.2.1 Implementation Results

The FPGA-based Gateway is developed using Verilog on Vivado version 2018.2. The implemented design is deployed on a Xilinx ALVEO U280 card, which includes UltraScale+ XCU280-L2FSVH2892E device (containing a total of 1,303,680 LUTs, 600,960 LUTRAM (LUTR), 2,607,360 FFs, 2,016 BRAMs and 9,024 DSP slices). In this implementation, HLS Pragmas (*HLS Dataflow*, *HLS Pipeline* and *HLS Unroll*) are used for optimisation. Table 6.1 shows the implemented blocks' hardware resource usage, timing and power consumption estimation.

Table 6.1: The implemented results of FPGA hardware on Vivado 2018.2.

Hardware Resources Usage					
Modules	**LUT**	**LUTR**	**FF**	**BRAM**	**DSP**
Input Abiter	2,944	1,112	5,980	0	0
Packet Processor	40,391	1,248	49,907	63	898
Output Queues	2,030	272	5,234	25.5	0
Gateway	156,056	15,606	228,901	294.5	898
(%)	11.97	2.60	8.78	14.61	9.95

Timing and Power				
F_max	**Hard IP**	**Dynamic**	**PL Static**	**Total**
309.8MHZ	1.38W	15.06W	3.64W	20.08W

The Gateway consumed 11.97% LUTs, 8.78% FF and 14.61% BRAM of the ALVEO U280 card. The *Packet Processor* block used 40,391 LUTs, which contributed approximately 25.9% to the Gateway resource usage. Besides, this block required 898 DSP slices, which contributed to 9.95% DSP available on the card, for ANN and CNN calculations. The maximum frequency is approximately 309.8MHz, and the total power is reported as 20.08W.

6.3.2.2 Performance

This test measures the processing throughput of the FPGA-based Gateway at high speeds on two criteria: processing speed and packet loss rate. This is a standalone test on the Security Admin host machine, utilising the Ethernet Card to replay network packets directly to the Gateway using QSFP28 transceivers. Network packets are divided into six groups based on the packet length (ranging from 64B to 1500B). For each group, the transmission rate is gradually increased (ten trials each) by using TCPreplay processes (AppNeta 2024). Figure 6.11 depicts the test results through six bar charts.

For packets with a length of 64B, the packet transmission rate is gradually increased from 0.20Gbps to 1.41Gbps. Before packet loss occurred, the maximum throughput recorded for forwarding raw packets and the ANN scanner were 1.12Gbps and 1.02Gbps, respectively. At the 10th trial, packets are replayed at 1.41Gbps; the packet loss rates in the two cases were 16.8% and 20.5%, respectively. With greater packet lengths, such as 128B, 256B, 512B, and 1024B, the Gateway achieved higher throughput rates before packet loss. In detail, the throughput is recorded at 2.1Gbps, 4.2Gbps, 8.3Gbps, and 14.0Gbps, respectively. The 1500B packets produced the most stable results, maintaining

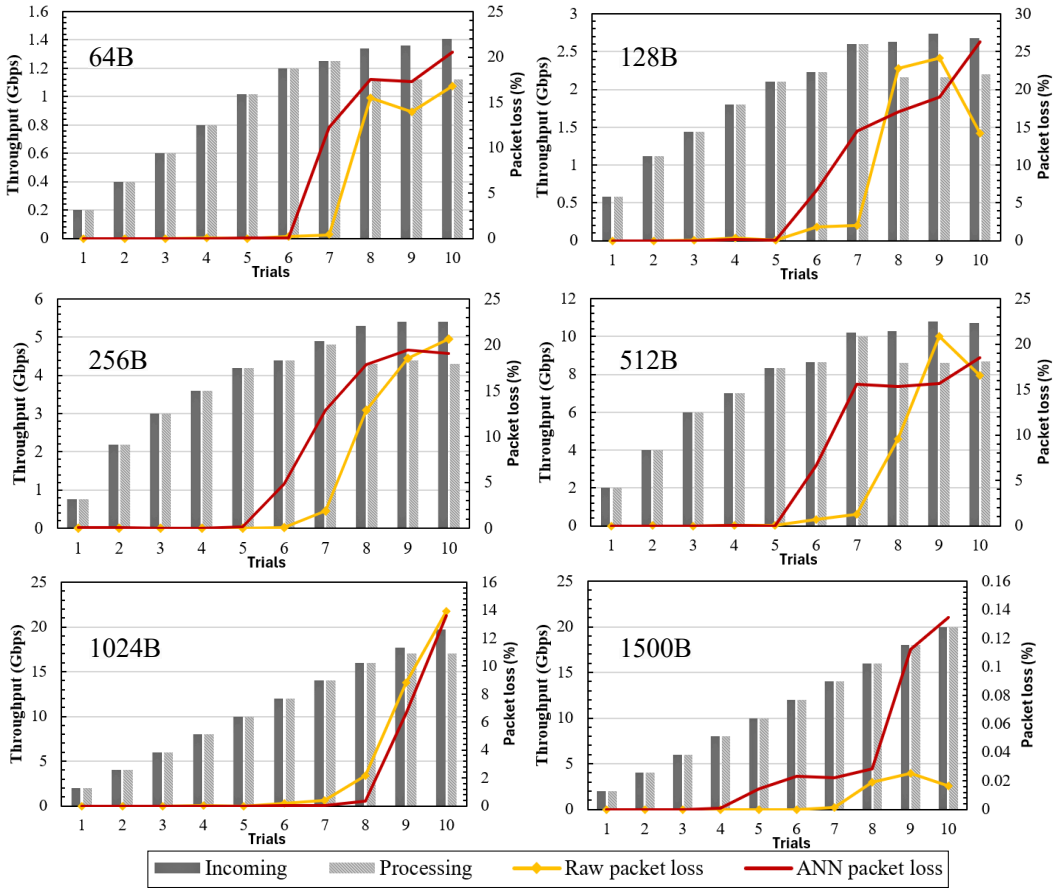


Figure 6.11: The performance test results for different packet sizes (64B-1500B).

an incoming speed of 20.0Gbps for forwarding raw packets and 0.85Gbps for generating ANN results, with corresponding packet loss rates of approximately 0.02% and 0.13%. The CNN model has produced a maximum processing speed of 8.38 Mbps during the tests, corresponding to 16,895 packets per second.

6.3.3 Smart Contract

6.3.3.1 Transaction Cost

The smart contract is compiled using the Remix IDE with compiler version 0.8.0. The contract's constructor function is initialized with the number of maximum admins and devices at 64 and 65,535, respectively. The *Simulator* generates transactions and interacts with the *Blockchain Server* through the *SSH Gateway*, as shown in Figure 6.10. The contract deployment transaction was successfully minted in block 1 with 3,025,154 gas units spent. Table 6.2 shows the average gas units spent and the transaction time (seconds) to perform the associated

contract function calls.

Table 6.2: Transaction time and gas used of functions in the smart contract.

Function	**Gas used**	**Transaction time (s)**
addAdmin	31,734	0.13
editAdmin	30,028	0.11
requestMonitor	127,876	0.09
monitorAccepted	32,428	0.05
editMonitoringDeviceList	44,901	0.07
addGatewayDevice	48,689	0.14
addEdgeDevice	48,670	0.07

Transaction time is measured when the Simulator creates a transaction and sends it over the internet via the *SSH Gateway* to the *Blockchain Server*. This time value fluctuates between 0.05 to 0.14 seconds. Regarding transaction cost, the *editAdmin* function requires the lowest gas fee, averaging around 30,028 gas units per transaction, while other functions consume between 31,734 and 48,689 gas units on average. The *requestMonitor* function needs at least 41,471 and can use up to 214,280 gas units, depending on whether the device address is at the beginning or the end of the device array being queried.

6.3.3.2 Device Verification

This test measures the device verification time on the smart contract. The number of admins in each trial (from 1 to 64 admins) is doubled. Each admin manages a maximum of 20 gateways and 50 edge devices, which are generated randomly. The test started at Block 606 and finished at Block 3,107, in which a total of 64 admins, 650 gateways, and 1,745 edge devices were added. The total gas units used is 121,200,664. Figure 6.12 shows the test results in detail. The horizontal axis presents the number of admins; the left and right vertical axes indicate the total number of devices and the verification time, respectively.

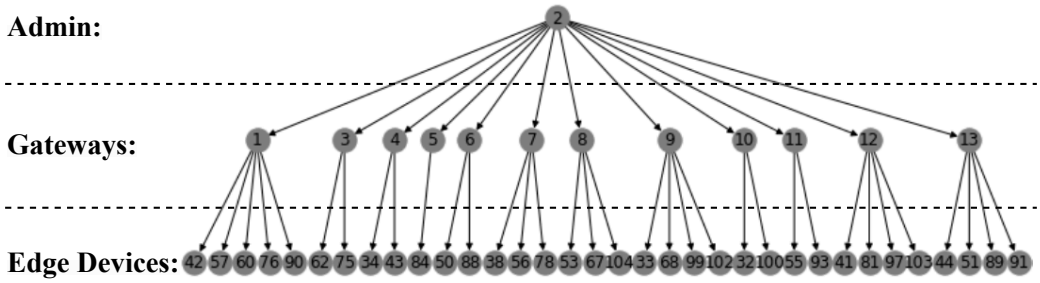


Figure 6.13: The connection topology generated from Blockchain data in the first scenario.

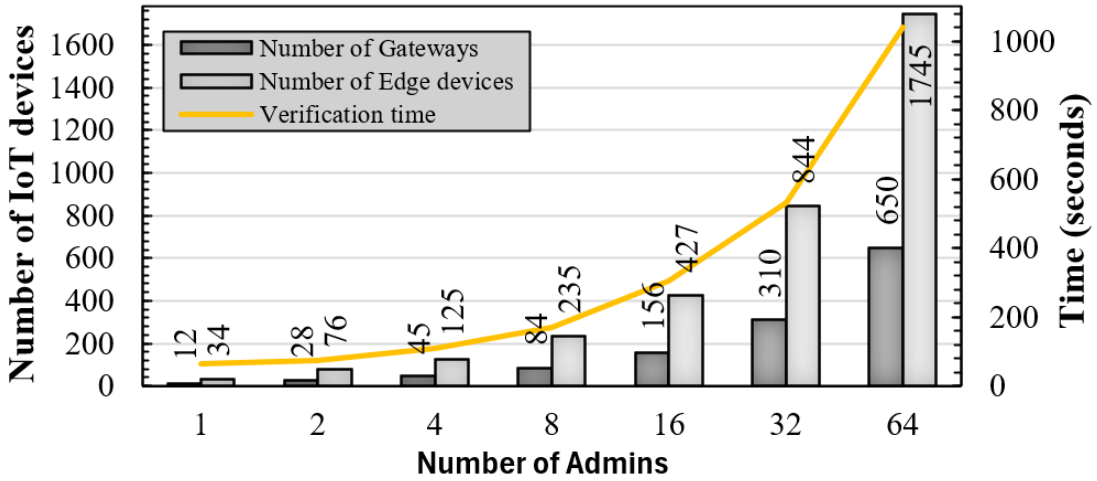


Figure 6.12: The device verification time on the smart contract in different scenarios.

At the beginning of the test, it takes 64 seconds to verify one admin and 46 devices. This number increased to 170 seconds when there were eight admins and 319 devices. The verification time linearly increased along with the total number of devices. In the last trial with 64 admins and 2,395 devices, the verification time is recorded as 1,040 seconds. Figure 6.13 illustrates a generated topology in the first trial where each number is mapped with an EVM address.

Next, the transaction time and the cost to store up to 65,535 new devices on the Blockchain are measured. The test started from block 3,108 and ended at block 68,642, with a total of 65,535 devices added. The test result is shown in Table 6.3.

Table 6.3: The total transaction time and gas cost when adding more devices.

Edge devices	Transaction time (s)	Total gas used
1,024	48.60	49,105,376
2,048	99.60	98,930,068
4,096	199.27	198,579,800
8,192	408.86	397,878,964
16,384	830.38	796,553,000
32,768	1,659.40	1,593,943,060
65,535	3,339.61	3,188,724,368

The transaction time (seconds) and gas units are recorded as 48.6 and 49,105,376, respectively, to add 1,024 devices. By doubling the number of new devices each time, the transaction time and the gas units are likely to be doubled. At the end of the test, after 65,535 devices are added, the corresponding transaction time (seconds) and transaction cost (gas units) are recorded as 3,339.61 and 3,188,724,368.

6.3.4 Platforms Comparison

This test compares inference time (processing time) between CPU (Intel(R) Xeon(R) Platinum 8276 CPU @ 2.2GHz x 112), GPU (NVIDIA Quadro GV100) and FPGA hardware (Alveo U280). The test scenario can be explained as follows:

1. Network packets in the IoT-23 dataset are selected for replaying back to the system. These packets belong to three benign scenarios, namely *CTU-Honeypot-Capture-4-1*, *CTU-Honeypot-Capture-5-1* and *CTU-Honeypot-Capture-7-1*. Table 6.4 briefly shows the scenario information, including the total file size and the number of packets.

Table 6.4: The pcap file's information.

Scenarios	Total size (MB)	Number of packets
CTU-Honeypot-Capture-4-1	7.0	30,189
CTU-Honeypot-Capture-5-1	381.4	398,312
CTU-Honeypot-Capture-7-1	151.9	553,039

2. The CPU and GPU in the *Blockchain Server* host machine are used in this

test. The network packets in the pcap format are read from the host machine storage and then fed to the inference model sequentially to simulate the real-time network incoming flow. The inference procedure is described in Section 6.1.3, which introduces pre-processing and processing time. The pre-processing time includes the CPU processing time for reading network packets and extracting input features. The processing time comprises data transfer and actual inference time.

3. The FPGA inference time is measured on the *Security Admin* host machine. The Ethernet Card is used to replay network packets at high speed. For testing the secure cores, a speed limit in replaying the packets is set; these limits are 2Gbps and 5Mbps for measuring ANN and CNN cores, respectively. Table 6.5 shows the routing information programmed on the FPGA Gateway in which raw packets will be forwarded to the physical interface - NF1, while security cores will send the results using virtual interfaces through the DMA route.

Table 6.5: The routing information programmed on the FPGA Gateway.

Packet types	Routing Ports
Raw network	NIC1
ANN alarm	DMA_OUT0
CNN alarm	DMA_OUT1

The experimental results for *CTU-Honeypot-Capture-4-1*, *CTU-Honeypot-Capture-5-1* and *CTU-Honeypot-Capture-7-1* scenarios are correspondingly illustrated in Figure 6.14, Figure 6.15 and Figure 6.16.

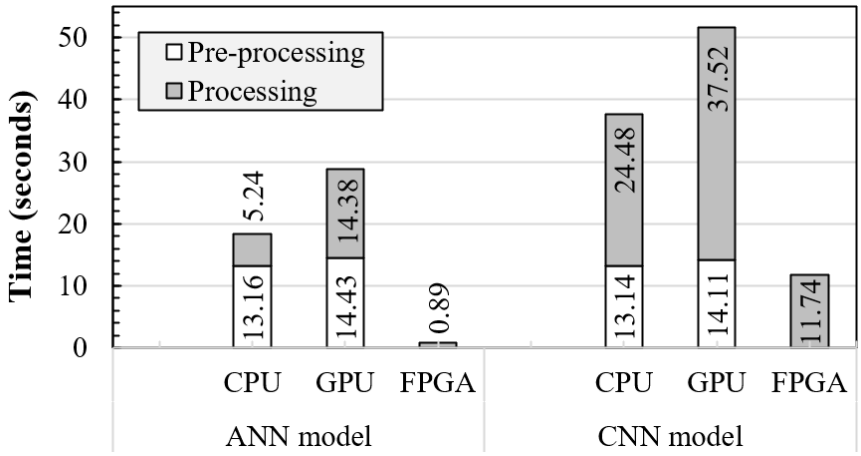


Figure 6.14: The inference results between engines in the first scenario.

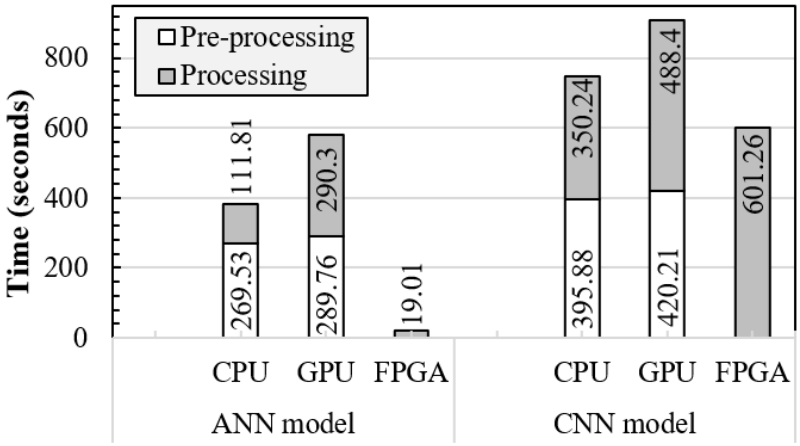


Figure 6.16: The inference results between engines in the third scenario.

Overall, the ANN model on FPGA shows an average speedup of 26.8 and 44.3 times that of the CPU and GPU used in this experiment, respectively. When the CNN model is used, the speedup is reduced to 1.8 and 2.2 times, respectively, compared to the ANN model.

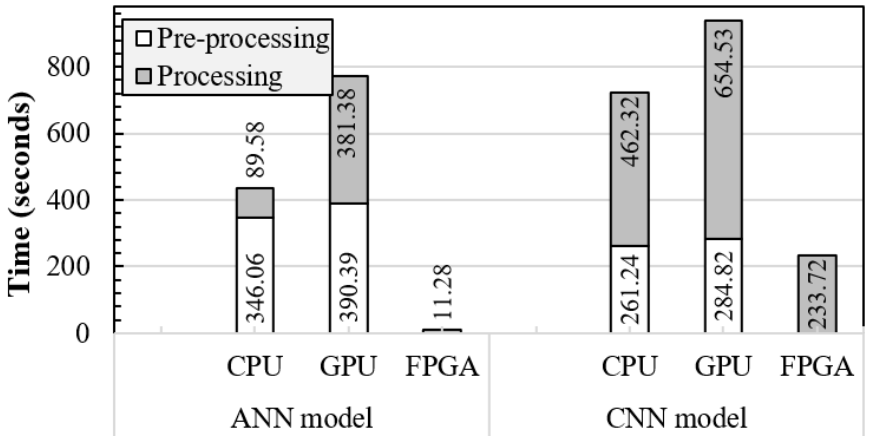


Figure 6.15: The inference results between engines in the second scenario.

In the first scenario, it takes 0.89 seconds for the FPGA Gateway to process 30,189 packets with an average packet size of 231B at the sending speed of 2Gbps. There is overhead in data pre-processing on the CPU that added approximately 251% and 100% to the total inference time of CPU and GPU, respectively. All scenarios have shown similar result patterns in which the FPGA card is the fastest in processing network packets. This is because the FPGA device used is specialised for networking applications. Additionally, data pre-processing is implemented on hardware using hardware description languages, while the CPU and GPU incur significant overhead in data pre-processing and

data transfer between memories. ML models on CPU and GPU platforms only achieve optimal performance when input data is bundled, which is not applicable in real-time packet processing.

6.3.5 FPGA Reliability

This section reports the reliability metric introduced in Section 6.3.3.2. The test was performed using packets from *CTU-Honeypot-Capture-4-1* (HC4-1), *CTU-Honeypot-Capture-5-1* (HC5-1), and *CTU-Honeypot-Capture-7-1* (HC7-1) scenarios, which were described in the previous section. The *test_small.pcap* file contains the first 100 packets in the *CTU-Honeypot-Capture-4-1* scenario. Table 6.6 summarised the reliability metrics of ANN and CNN models on the Alveo U280 FPGA.

Table 6.6: FPGA Reliability results of ANN and CNN models.

Scenarios	**Files**	**No. of packets**	**Packets Match**	
			ANN model	**CNN model**
HC4-1	2018-09-14-13-40-25-Philips-Hue-Bridge	8,573	8,573 *(100%)*	8,567 *(99.93%)*
	2018-10-25-14-06-32-192.168.1.132	21,664	21,664 *(100%)*	21,660 *(99.98%)*
HC5-1	2018-09-21-capture	398,312	398,312 *(100%)*	398,127 *(99.95%)*
HC7-1	2019-07-03-15-15-47-first_start_somfy_gateway	8,276	8,276 *(100%)*	8,276 *(100%)*
	2019-07-04-16-41-10-192.168.1.158	62,850	62,850 *(100%)*	62,630 *(99.97%)*
	2019-07-07-16-41-19-192.168.1.158	119,158	119,158 *(100%)*	119,114 *(99.96%)*
	2019-07-03-16-41-09-192.168.1.158	119,638	119,638 *(100%)*	119,612 *(99.98%)*
	2019-07-05-16-41-14-192.168.1.158	121,554	121,554 *(100%)*	121,460 *(99.92%)*
	2019-07-06-16-41-17-192.168.1.158	121,566	121,566 *(100%)*	121,520 *(99.96%)*
	Total:	**981,591**	**981,591 (100%)**	**981,166 (99.96%)**

There are nine network files with a total of 981,591 network packets, which were transmitted to the FPGA card. The ANN model on the FPGA card produced fully matching results to the ANN model on CPUs, achieving a reliability of 100%. In contrast, the CNN model demonstrated a slight discrepancy in computations on the FPGA card compared to CPUs. When using the CPU-based

CNN model as the reference, the reliability of the CNN model on FPGA was measured at 99.96%.

6.4 Discussion

The proposed architecture includes security hardening at different layers, enhancing protection from the edge to the cloud of an IoT network. This section discusses the advantages and disadvantages of the implemented system by layers as follows.

6.4.1 The Cloud Layer

Blockchain and smart contract technology are used for access control in IoT networks on the cloud. Blockchain is well-suited for IoT security as organisations increasingly adopt decentralised network models. In a decentralised platform, widespread adoption means more validators participating in the network, strengthening the system's security and sustainability. This enhances data integrity and redundancy in IoT networks.

The IoT network generates a significant amount of data over time; thus, recording this data on the Blockchain is costly (the gas used to perform a transaction on the Blockchain). This work proposes smart contracts to store management data on-chain, including administrators, devices and connections information. In addition, using smart contracts helps prevent attackers from impersonating devices or administrators since only whitelisted participants can change the smart contract's states.

The contract owner holds the highest authority level, which is vulnerable if compromised. This security threat can be mitigated using the multi-signature technique, in which a group of administrators replaces the owner, and the majority must approve each function call in the smart contract before execution. Furthermore, a smart contract can be updated without changing the original contract address using the proxy technique. This enhances flexibility and security since the smart contract is upgradeable.

6.4.2 The Control Layer

Smart contracts allow administrators to operate in a distributed manner while maintaining data synchronisation. Approved Web3 admins can quickly query

IoT network data and participate in network governance. Similarly, security admins can contribute equally through smart contracts. This approach enhances the system's scalability in a cost-effective manner. However, there is a collision when different admins make different decisions on the same task, particularly if one of the admins has compromised their private key. In such cases, an action is required to detect and remove the compromised admin from the admin list.

At the current stage, an administrator must supervise and add new devices to the smart contract after monitoring results from the anomaly-based IDS. A better handshaking mechanism between the Blockchain and anomaly-based IDS is needed for an automatic system.

6.4.3 The IoT Devices Layer

Although the ANN model has shown outstanding performance, thanks to the pipeline design on FPGA, the CNN model has experienced overhead in feature extraction and post-processing. The CNN model performance can be improved by either redesigning the CNN model on FPGA or changing the priority in the *Packet Selector* module.

The Gateways are designed on programmable FPGA-based hardware accelerators, resulting in enhanced flexibility in designing and updating security functions. However, the Alveo U280 device used in this study demonstrates high-speed processing capability; it is a high-end device with high production costs, which may pose challenges for practical applications.

The edge devices in this study are currently in a simulated state, and a practical system would encounter additional challenges. For instance, most traditional devices are limited in computational power, making it impractical to manage the EVM key for Blockchain authentication. Consequently, the administrators in the control layer must ensure that the mapping process is accurate and secure.

6.5 Summary

This chapter presented the implementation of the scalable security approach that combines Blockchain and anomaly-based IDS using hardware accelerators for IoT networks. The implemented system has demonstrated the comprehensive protection approach from the edge to the cloud in an IoT architecture.

Blockchain and smart contracts are proposed for IoT access control on the

cloud. Blockchain provides a transparent, immutable, and decentralised computing environment in which administrators can control IoT devices in the network. The smart contract includes two authorisation levels for managing administrators and IoT devices. The smart contract's transaction costs and response time in different test cases are measured. The experimental results showed that the implemented system effectively manages administrators and devices according to defined rules in the smart contract, in which only authorised entities can access the system. Future work will explore adding security features for smart contracts, such as multi-signature and proxy access.

Secondly, an anomaly-based IDS for edge devices is proposed. The proposed approach uses lightweight machine learning models on FPGA hardware accelerators for Gateways. The prototype FPGA-based Gateway, which contains ANN and CNN detection models, was deployed on the Alveo U280 card (NetFPGA-PLUS) to evaluate the proposed architecture. The ANN model has achieved the highest processing speed at 20Gbps, with a packet loss rate of approximately 0.13%. The CNN model has produced a maximum processing speed of 8.38 Mbps, corresponding to 16,895 packets per second. The CNN architecture on FPGA can be redesigned in a pipeline fashion for better performance in the next version.

Finally, the chapter's results indicate that combining Blockchain and anomaly-based IDS significantly enhances scalable security in IoT networks. The anomaly-based IDS can detect attacks early at the edge level using hardware accelerators, while the smart contracts on Blockchain help manage and scale the system effectively in a secured and decentralised manner.

References

AMD (2021). *AMD ALVEO U280: Adaptable Accelerator Cards for Data Center Workloads*. Accessed: Dec 16, 2024.

AppNeta (2024). *Tcpreplay*. https://tcpreplay.appneta.com/ visited on Dec 16, 2024.

Garcia, S, A Parmisano, and MJ Erquiaga (2020b). "IoT-23: A labeled dataset with malicious and benign IoT network traffic". In: *Stratosphere Lab., Praha, Czech Republic, Tech. Rep*.

NetFPGA (2021). *NetFPGA PLUS*. https://netfpga.org/NetFPGA-PLUS.html visited on Dec 16, 2024.

Ngo, Duc-Minh, Dominic Lightbody, Andriy Temko, Cuong Pham-Quoc, et al. (2023b). "Network Attack Detection on IoT Devices Using 2D-CNN Models". In: *International Conference on Intelligence of Things*. Springer, pp. 237–247.

Chapter 7

Conclusion and Future Work

The thesis studies scalable security approaches for IoT networks utilising distributed-ledger technology and machine learning on hardware accelerators. This chapter concludes the thesis by summarising the research findings and discussing future work.

Chapter 2 studies signature-based and anomaly-based IDSs for detecting network attacks.

- The proposed anomaly-based IDS framework uses lightweight ANN models to detect attacks in IoT networks. The framework includes dataset pre-processing, GPU-based training acceleration, and offline inference using hierarchical decision-making.
- The proposed algorithms are tested with the new IoT-23 dataset using ANN models and achieved the highest accuracy of 99.68%. The accuracy of detection is increased further to 99.79% by applying the hierarchical decision-making method.
- Research findings have indicated that the vast majority of anomaly-based IDSs have been focused on improving the model's detection accuracy, thus increasing the model complexity and overhead in the inference phase. One solution to tackle these issues is using hardware accelerators. Networking systems experience varying traffic volumes depending on events, with peak loads or flooding attacks that can result in high-volume traffic; thus, anomaly-based IDS should be capable of handling the high load to mitigate attacks.
- An ANN model's inference process is straightforward, and the parameters

in the models can be quantised and converted into integers, which aligns well with FPGA hardware architecture.

Chapter 3 leverages the proposed framework using hardware acceleration solutions to enhance IoT security at the edge. The proposed framework is tested with the UNSW-NB15 and IoT-23 datasets using ANN models. The inference phase is implemented and tested using different approaches on the CPU, GPU, AI-assisted microcontroller and SoC FPGA platforms. In addition, the inference phase on SoC FPGA is implemented using HLS and Verilog approaches, achieving the same practical processing speed. However, the pipelined design on the PYNQ-Z2 board with the Xilinx Zynq xc7z020-1clg400c device has been optimised to run at the on-chip frequency with a fixed 28 clock cycles overhead.

Chapter 4 studies the deep packet inspection approach using deep neural network models.

- The CNN model is used for processing network packets, and the feature extraction module has been erased.
- The proposed approach involves labelling network packets in the IoT-23 dataset from pre-trained ANN models; then, CNN models are trained to detect attacks at the packet level.
- By utilising the CNN model, all data in the packet is treated as valuable in contributing to the final decision. This approach also implies that accurate labelling of each packet is crucial during training processes.
- The experimental results show that the best model has achieved a comparable accuracy of 99.34% on the IoT-23 datasets. More importantly, this approach has mitigated the computational overhead of the feature extraction process in the inference phase, resulting in an approximately 40% improvement in feature extraction time. The proposed approach is experimented with different CNN models to optimise its architecture for implementing hardware acceleration platforms.

Chapter 6 presents the proposed scalable security approaches that combine Blockchain and anomaly-based IDS using hardware accelerators for IoT networks. The implemented system has demonstrated the comprehensive protection approach from the edge to the cloud in an IoT architecture.

- Blockchain and smart contracts are proposed for IoT access control on the cloud. Blockchain provides a transparent, immutable, and decentralised

computing environment in which administrators can control IoT devices in the network.

- The prototype FPGA-based Gateway, which contains ANN and CNN detection models, was implemented on the Alveo U280 card (NetFPGA-PLUS) for high-speed inference.

The scalable security approach, which utilises Blockchain and anomaly-based IDS, is promising in IoT networks. The anomaly-based IDS can handle high-speed network traffic and detect attacks early at the edge using hardware accelerators, while the smart contracts on Blockchain help manage and scale the system effectively in a secured and decentralised manner.

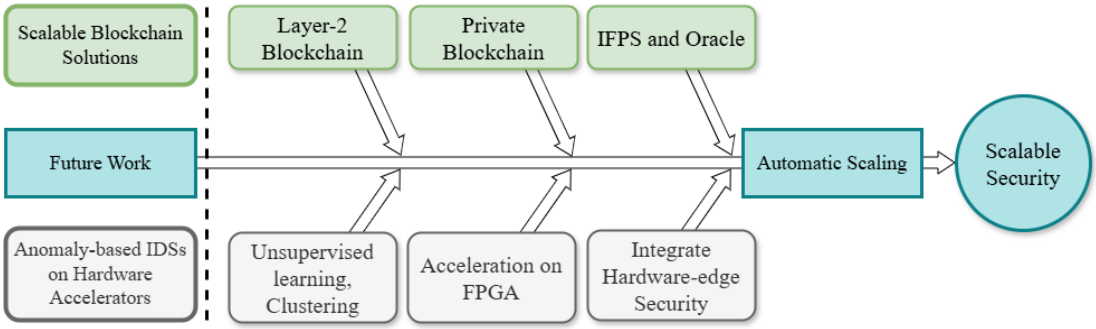


Figure 7.1: Future work on enhancing scalable security in the proposed system.

Future work includes hardening scalable Blockchain solutions and anomaly-based IDSs on hardware accelerators. Figure 7.1 summarises the future work that will be discussed below:

7.1 Scalable Blockchain Solutions

The smart contract is proposed and tested on a simulated Blockchain (Ganache). To enable the advantages that a Blockchain offers, the system must be deployed on a Blockchain mainnet, where data and smart contracts are genuinely synchronised and executed in a decentralised manner. Additionally, scalable Blockchain solutions involve improving transaction throughput, optimising data storage, and reducing transaction costs.

- **Exploring layer-2 Blockchains (permissionless)**: Ethereum-based layer-2 Blockchains offer better transaction throughput and lower transaction cost than the Ethereum Blockchain by using rollups technology to reach consensus. However, these Blockchains incentivise miners by

distributing the collected digital assets (Ethereum) from transactions as mining rewards. It would still become prohibitively expensive for large-scale IoT networks to write data on the Blockchain.

- **Smart contracts on private Blockchains (permissioned)**: Hyperledger Blockchains will be used to manage IoT networks. A private Blockchain would suit organisations that require a higher level of flexibility in operating the Blockchain to some extent; however, they compensate for the security, integrity, and redundancy compared to public Blockchains.
- **Interplanetary File Systems (IPFS) and Oracle technologies for data storage**: Since IPFS offers a decentralised methodology to store large amounts of data in a distributed network, reducing the reliance on a centralised server; at the same time, Oracles technology fetches and verifies this data, ensuring that it is accurate before being used by smart contracts. Combining the two technologies creates a decentralised, efficient, and secure system for storing, retrieving, and validating off-chain data in Blockchain applications, enabling tamper-proof and scalable solutions for smart contracts.

7.2 Anomaly-based IDSs on Hardware Accelerators

With the rapid proliferation and development of AI, anomaly-based IDSs are continuously evolving with improved algorithms. However, effectively applying these systems to IoT networks remains a critical topic that requires researchers' attention and testing efforts. Below are the primary directions for improving the proposed anomaly-based IDSs.

- **Approaching unsupervised learning methodologies**: This thesis has indicated that supervised learning techniques rely heavily on the accuracy of data labelling prior to the training phase. For a system to operate sustainably, it must meet two additional criteria:
  - The independent analysis ability in which the system is able to classify data without the label information. Clustering techniques can solve the issue by grouping data based on its similarity.
  - Collaborative and Continuous Learning: Federated Learning (FL) ad-

dresses this requirement by enabling distributed systems to update machine learning models iteratively and collaboratively. Furthermore, FL is a decentralized approach that complements Blockchain technology as both rely on principles of decentralisation and distributed systems. They can enhance data privacy, scalability, and security to each other.

- **Optimise and accelerate the design on FPGA**: Along with improving ML models, there is a need for mapping and accelerating it on hardware accelerators. The thesis tested the proposed system on a high-end FPGA device (Alveo-U280) specialized for high-speed networking applications. There are unused hardware resources for integrating and expanding security functions. Future work will accelerate different ML algorithms on this platform, focusing on high-speed and low-power consumption designs. Besides, when approaching hardware design, the design verification process is critical to ensure all the functions are secured and accurate. The Universal Verification Methodology (UVM) (Siemens 2025) can be applied to validate and simulate the proposed design on FPGA.
- **Integrate hardware attack detection at edge devices**: Side-channel attacks are investigated by other members in the group. The power data in the previous work will be studied to build a system for detecting and classifying attacks in the proposed framework.

In addition, future work will improve the system's automatic scaling ability when combining anomaly-based IDSs and Blockchain to enhance scalable security in IoT networks. This will include studying and experimenting with different prototype systems to improve the communication between the edge and the cloud.

References

Siemens (2025). *UVM - Universal Verification Methodology*. `https://verificationacademy.com/topics/uvm-universal-verification-methodology/`. Accessed: 2025-03-07.