



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

TRINITY COLLEGE DUBLIN

THE UNIVERSITY OF DUBLIN

DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING

On Enabling Intrusion Detection Systems for Automotive Networks

Author:

Shashwat Khandelwal

Supervisor:

Dr. Shreejith Shanker

Submitted
April 14, 2025

I dedicate this thesis to my family, partner and friends who supported me during this challenging journey.

Declaration

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the Library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

Signed:

Shashwat Khandelwal

Shashwat Khandelwal
August 2025

Summary

The rapid evolution of software-driven functionalities in modern vehicles has led to the availability of advanced driving assistance systems. In tandem with the above developments, the increasing number of sensors in vehicles to support such new features has led to a significant rise in the number of in-vehicle electronic control units (ECUs), network interfaces, and communication protocols. This growing complexity enhances vehicle performance, connectivity, and comfort applications. The increased connectivity to the outside world, however, introduces new security challenges. As automotive networks become more interconnected, they are exposed to a wider array of cyber threats that endanger the safety of passengers as well as threaten the loss of their personal and private information. This necessitates the inclusion of robust Intrusion Detection Systems (IDS) in vehicular networks to safeguard vehicle communications. Given the importance of the problem, multiple IDSs have been proposed in the research literature in the past 10 years. However, conventional IDS implementations often impose significant computational overhead, requiring additional ECUs and network elements that can contribute to increased system cost, complexity, and power consumption.

In this thesis, the feasibility of consolidating intrusion detection capabilities within critical ECUs is explored to establish an efficient, embedded IDS architecture for modern automotive networks. By integrating IDS functionalities into hybrid ECU architectures, the aim is to mitigate the need for additional hardware peripherals utilised as IDSs while maintaining a robust approach to network security. This thesis focuses on the widely used Controller Area Network (CAN) protocol, which serves as the backbone of in-vehicle communication, enabling real-time and reliable data exchange among ECUs. To enhance network security, approaches are proposed that integrate quantised machine learning-based IDS models as lightweight, isolated security modules within ECU hardware. This minimises disruptions to standard software tasks while ensuring efficient per-message handling for real-time detection.

To evaluate the feasibility of this approach, three key aspects have been investigated: (1) whether IDS models can be lightweight enough for embedded deployment, (2) their generalisability across multiple datasets and unseen attack types, and (3) their ability to support per-message detection at line rates. The proposed IDS models are trained to detect a diverse range of cyber threats, including Denial-of-Service (DoS), Fuzzing, and Spoofing attacks, which pose significant risks to automotive networks. The experimental results demonstrate that the proposed IDSs achieve high detection accuracy across multiple attack vectors while maintaining low per-message processing latency and minimal energy consumption. This ensures compliance with the stringent constraints of real-time automotive environments, making embedded deployment of IDSs within a vehicle's network both practical and efficient. By leveraging the consolidation of IDS capabilities within critical ECUs, my research contributes to the advancement of secure and resilient automotive network architectures, paving the way for enhanced cybersecurity in next-generation vehicles.

Table of contents

List of figures	vii
List of tables	xi
1 Introduction	1
1.1 Evolution of Vehicular networks	1
1.1.1 Early Developments and the Introduction of ECUs	1
1.1.2 Emergence of Intra-Vehicular Networks: The CAN Protocol	2
1.1.3 Integration of Infotainment and Connectivity Systems	3
1.1.4 Challenges in Security and Cybersecurity Vulnerabilities	4
1.2 Motivation: Efficient & Scalable IDS for Automotive E/E	4
1.2.1 Computational Complexity	6
1.2.2 Generalisability and Evolving attack vectors	7
1.2.3 Integration Challenges	8
1.3 Research Questions	9
1.4 Contributions	9
1.5 Thesis Organisation	11
1.6 Publications & Open-source contributions.	12
2 Background	15
2.1 Automotive Electronic Control Units	15
2.2 In-vehicle networks	18
2.2.1 Controller Area Network	19
2.2.2 Automotive Ethernet	22
2.2.3 Other networks: FlexRay, LIN, MOST	24
2.2.3.1 FlexRay	24
2.2.3.2 Local Interconnect Network	25
2.2.3.3 Media Oriented Systems Transport	26

2.3	Security challenges in automotive networks	27
2.4	Intrusion detection systems for CAN	33
2.4.1	Early Intrusion Detection Systems (2015-2020)	34
2.4.2	Machine Learning-based IDSs for CAN	36
2.4.2.1	Public attack datasets	37
2.5	Deployment Challenges for IDSs	39
2.5.1	Understanding Real-Time Constraints in CAN Bus	39
2.5.2	Energy & Resource Constraints for Real-Time IDS in Auto- motive Networks	41
2.5.2.1	Energy Constraints for IDS Deployment	41
2.5.2.2	Resource Constraints for IDS Deployment	42
2.6	Field Programmable Gate Arrays	43
2.6.1	Machine Learning on FPGAs	43
2.6.2	FPGAs for automotive applications	46
2.7	Summary	47
3	Quantised feed-forward NN as IDSs for Controller Area Network	49
3.1	Introduction & Contributions	49
3.2	Baseline Model: Quantised deep-CNN	50
3.2.1	CNN-based IDS	51
3.2.2	Dataset and Training	51
3.2.3	IDS Integration Architecture: QdCNN & MA-QCNN	52
3.2.3.1	Mapping models to the DPU	52
3.2.3.2	Integration with Zynq Ultrascale+	53
3.2.4	Experimental evaluation	54
3.2.5	Accuracy	55
3.2.6	Inference Latency, Power & Resource consumption	56
3.2.7	Performance scaling using alternate DPU configurations	60
3.3	QdCNN optimisation: Transfer learning & Pruning	62
3.3.1	MA-QCNN model	62
3.3.2	Dataset & Training	62
3.3.3	Experimental Evaluation	63
3.3.4	Accuracy	63
3.3.5	Inference Accuracy	64
3.3.6	Latency, Power consumption & Resource utilisation	65
3.4	Quantised MLPs as IDSs	67

3.4.1	MLP Model	68
3.4.2	Dataset & Training	68
3.4.3	IDS Integration Architecture: QMLPs	69
3.4.4	Experimental Evaluation	69
3.4.5	Accuracy	71
3.4.6	Inference latency	73
3.4.7	Power consumption, PS/PL utilisation	75
3.5	Summary of DPU-accelerated quantised models	77
3.6	Custom quantised MLPs as IDSs : CQMLP	78
3.6.1	Model Architecture	78
3.6.2	Design space exploration	79
3.6.3	Dataset and Training	81
3.6.4	Dataflow hardware generation and integration to ECU-IDS . .	82
3.6.5	Experimental Evaluation	83
3.6.5.1	Inference accuracy	83
3.6.5.2	Inference latency	85
3.6.5.3	Power consumption, Resource Utilisation (PS/PL) .	86
3.7	Summary	86
3.8	Related Publications	87
4	Zero-Day Attack Detection on CAN networks	89
4.1	Introduction & Contribution	89
4.2	Unsupervised learning models as IDSs	91
4.3	Autoencoders as anomaly detectors	93
4.4	System architecture	95
4.4.1	CAE model as unsupervised IDS	95
4.4.2	Design space exploration for CAE	96
4.4.3	Dataset and Training	98
4.4.4	Integration Architecture	101
4.5	Experimental Results	101
4.5.1	Accuracy	103
4.5.2	Inference latency	105
4.5.3	Power consumption and Resource Utilisation	107
4.6	Summary	109
4.7	Related Publications	109

5	Quantised Recurrent Neural Network based IDS	111
5.1	Introduction & Contributions	111
5.2	QLSTM layer development	113
5.2.1	Recurrent Neural Networks	113
5.2.2	Software front end	114
5.2.3	FINN-Compiler Optimisations	115
5.2.4	QLSTM HLS Backend	117
5.3	QLSTM-IDS	119
5.3.1	Model Development	119
5.3.2	Dataset, Training & Deployment	121
5.3.2.1	Dataset Descriptions:	121
5.3.2.2	Training procedure	122
5.3.2.3	Generating the dataflow accelerator	124
5.3.3	Bridged IDS Integration	126
5.4	Experimental Results	128
5.4.1	Accuracy	128
5.4.2	Detection Latency	135
5.4.3	Energy & Resource Utilisation	137
5.4.4	Discussion: QNNs proposed as IDSs	138
5.5	Summary	140
5.6	Related Publications	141
6	SecCAN: CAN Controllers with Embedded Intrusion Detection	143
6.1	Introduction & Contributions	143
6.2	Integration Strategies	146
6.3	SecCAN: IDS integration	148
6.3.1	Extending Controller Datapath for IDS	148
6.3.2	Model & Dataset for training and testing	150
6.4	Evaluation & Results	150
6.4.1	Accuracy	152
6.4.2	Detection Latency	152
6.4.3	Resource & Energy Consumption	154
6.5	Summary	155
6.6	Related Publications	156

7	Summary & Future Research	157
7.1	Summary of Contributions	157
7.1.1	Motivation and Initial Exploration	157
7.2	CNN-Based IDSs and Their Limitations	157
7.3	MLP-Based IDSs for Improved Efficiency	158
7.4	Unsupervised Learning for Zero-Day Attack Detection	159
7.5	Capturing time-series signature using LSTMs	159
7.6	Integration of IDS in CAN Network Controllers	160
7.7	Concluding remarks	160
7.8	Future Research	161
7.8.1	Real-Time Intrusion Prevention and Mitigation Strategies	161
7.8.2	Integrated and Holistic Cybersecurity Frameworks	162
7.8.3	Dataset Expansion and Model Generalisation	162
7.8.4	Federated Learning for Distributed Threat Detection	162
7.8.5	Threat Intelligence Propagation and Knowledge Sharing	163
7.8.6	Adaptation to Emerging E/E Architectures	163
7.8.7	Concluding Remarks	163
	References	165
	Bibliography	178

List of figures

2.1	The figure depicts the internal and external structure of a typical ECU.	16
2.2	The figure illustrates the shared bus-based topology of CAN in in-vehicle networks.	20
2.3	Frame format of an extended frame CAN message. In the case of a standard CAN message, the ID field contains the 11-bit identifier. . .	21
2.4	Frame format of an error frame CAN message.	21
2.5	State diagram for error handling in CAN.	22
2.6	The figure illustrates the star topology of Automotive Ethernet in in-vehicle networks where nodes communicate with each other via Ethernet switches, effectively communicating like a point-to-point connection.	23
2.7	System diagram of the CAN network (featuring components connected to high-speed/low-speed CAN) shows how an affected ECU can cause network tampering.	32
3.1	The proposed IDS-ECU architecture with the DPU accelerator attached as a standard peripheral to the processor	55
3.2	Per-message latency of the QdCNN model on all DPU configurations after enabling multiple threads from 1 to 3.	61
3.3	ROC curve of the MA-QCNN model on the DoS and Fuzzing attacks.	65
3.4	Proposed system architecture of the IDS-ECU with the quantised MLP accelerators on the PL part of the FPGA.	70
3.5	ROC curve of the MA-QMLPs on the four attacks.	74
3.6	Latency improvement observed with increased model optimisation from QdCNN to QMLP.	78
3.7	The proposed MLP model as a multi-attack detection IDS.	79
3.8	Training loss of the model for different precision of weights and activations for all the attacks.	80

3.9	Validation loss of the model for different precision of weights and activations for all the attacks.	81
3.10	The proposed IDS-ECU architecture. The ML model is accelerated on the PL part of the FPGA.	82
4.1	An illustration of a zero-day attack launched through a compromised ECU left undetected by an IDS. The top figure shows a supervised learning (SL) based IDS flagging the known DoS attack on the CAN bus, while in the bottom figure, the IDS is unable to flag the attack on the CAN bus as the compromised ECU uses zero-day attack techniques.	90
4.2	The proposed convolutional autoencoder model architecture as a zero-day attack detection IDS. ‘n’ denotes the window size of the input feature used for attack detection. The proposed model uses $n = 100$ in this chapter.	97
4.3	The figure illustrates the performance of the proposed models on the attack-free portions of the datasets, which are subsequently used to determine the attack detection thresholds.	99
4.4	The figure illustrates the accuracy of the proposed models in detecting unseen attacks from the CH and SA datasets, using thresholds derived from testing on their respective benign subsets.	100
4.5	Proposed system architecture of the integrated QCAE-IDS. The quantised CAE model is accelerated on the PL part of the FPGA device with the ping-pong buffers acting as the pre-processing block for the accelerator.	102
5.1	The figure illustrates how the Scan operator replicates the recurrent hidden state computations in an LSTM layer. The yellow, red, and blue arrows highlight the pathways for input data processing and state updates at each time step for every input in the sequence.	115
5.2	The figure illustrates how the Multithresholding operator approximates the tanh activation function over a defined input range for INT2, INT4 and INT6 bitwidths.	117
5.3	The figure illustrates how the proposed transformations allow the floating point operations to be absorbed into the nearest multi-thresholding operator.	118

5.4	This figure illustrates the different stages in the overall deployment workflow for generalised quantised LSTM layer (with mixed quantisation support) based models as an extension of the FINN framework for FPGAs. This figure is inspired by Figure 6 in [147].	120
5.5	The figure shows the network architecture of the proposed LSTM-IDS. The CAN input is packed into 20 uint4 values as the input to the LSTM layer. The classified outputs are packed into a byte-wide data.	121
5.6	The figure illustrates the average training and validation loss curves for the QLSTM-IDS during its training for different activation precisions using the car hacking dataset. To better highlight the differences, the figure provides a magnified view of epochs 5 to 50.	123
5.7	The figure illustrates the average training and validation loss curves for the QLSTM-IDS during its training on the four attack datasets from the car hacking and survival analysis datasets.	125
5.8	The figure illustrates the ECU architecture modelled on a Zynq Ultra-scale+ hybrid SoC and the path of a CAN message upon reception in this architecture. In this architecture, the processing system (PS) functions as the ECU, executing automotive applications. The blue arrows indicate the path taken by the CAN message from the CAN bus to the software tasks on the PS. The IDS processing is integrated within the programmable logic (PL) section of the XCZU7EV device. The IDS can receive the CAN message directly from the controller (at the same instance as it is read by the PS) or forwarded by the PS. A multiplexer is used to control this at run-time, with the select line connected to an external peripheral on the board.	127
5.9	Confusion matrix capturing the classification results of the QLSTM-IDS for the Car Hacking Dataset. The figure also shows the success/failure rates for individual attacks as well as overall performance numbers.	131
5.10	Confusion matrix for QLSTM-IDS classification results on HYUNDAI YF Sonata 2010 Model attack set of SA dataset.	132
5.11	Confusion matrix for QLSTM-IDS classification results on KIA compact SUV Soul 2015 Model attack set of SA dataset.	132
5.12	Confusion matrix for QLSTM-IDS classification results on CHEVROLET mini-compact vehicle Spark 2015 Model attack set of SA dataset.	133

- 6.1 The figure illustrates conventional integration and placement strategies for CAN intrusion detection systems presented in the literature, alongside the proposed case for embedding IDS within the controller. 144
- 6.2 The figure illustrates the integration of the IDS within the CAN controller. The red arrows show the standard datapath within the controller with the purple arrows indicating the augmented path for IDS. 148
- 6.3 The waveform shows the signalling within the SecCAN controller for a 5-byte CAN message from the bus. The IDS operation overlaps with the protocol checks on the bus and is completed before the frame is ready to be read by the ECU. 151

List of tables

2.1	The table displays the various vehicle types along with the corresponding number of ECUs in each vehicle.	17
2.2	The table outlines the different vehicle types and the main functions performed by the ECUs in each vehicle.	17
2.3	The table provides the estimated bandwidth requirements for various sensors commonly used in in-vehicle networks.	22
2.4	Input features used by the ML-based IDSs & IPSs proposed in the research literature.	37
2.5	The table shows different CAN bus speeds and the corresponding maximum permissible IDS processing latency.	41
2.6	The table shows different ECU designs and their respective energy budgets.	42
2.7	The table shows different ECU types and the corresponding resource budgets (compute performance, memory) available with them.	42
3.1	An extract from the open Car hacking dataset which is used for the testing and evaluation.	52
3.2	Inference accuracy metrics of the model pre- and post-quantization on the four datasets	56
3.3	Accuracy metric comparison of the QdCNN FPGA accelerator to other state-of-the-art methods	57
3.4	Confusion matrix capturing the classification results of the QdCNN on the DoS, Fuzzy, and Spoofing attacks	57
3.5	Per-message latency and measured power of QdCNN accelerator on different platforms.	58
3.6	Per-message latency comparison against state-of-the-art IDSs (GPU/Raspberry Pi) reported in literature	59

3.7	Resource utilisation breakdown for the PL modules of the proposed CAN IDS-ECU (XCZU7EV)	59
3.8	Performance scaling of the QdCNN model using different DPU configurations: impact on per message latency (ms), resource (%), & power consumption (W)	60
3.9	Inference accuracy metrics of the proposed MA-QCNN model, pre- and post-quantisation on the DoS & Fuzzing attacks.	63
3.10	Confusion matrix capturing the classification results of the MA-QCNN (on DPU).	64
3.11	Accuracy metric comparison (%) of the quantised FPGA accelerator (MA-QCNN) against the reported literature.	65
3.12	Per-message latency comparison against state-of-the-art IDSs (GPU/Raspberry Pi) reported in the literature.	67
3.13	Resource utilisation breakdown for PL accelerator of the proposed CAN IDS (XCZU3EG).	67
3.14	Inference accuracy metrics of QMLP-1, pre and post quantisation {Before and After Fine-tuning (BF & AF respectively)} on the DoS and Fuzzing attacks.	71
3.15	Inference accuracy metrics of QMLP-2 pre- and post-quantisation on the RPM & Gear attacks.	71
3.16	Accuracy metric comparison (%) of the quantised FPGA accelerator (QMLP-1) against the reported literature on the DoS and Fuzzing attacks.	72
3.17	Accuracy metric comparison (%) of the quantised FPGA accelerator (QMLP-2) against the reported literature on the RPM and Gear attacks.	73
3.18	Confusion matrix capturing the classification results of the QMLP-1 model on the DoS and Fuzzing attacks.	74
3.19	Confusion matrix capturing the classification results of the QMLP-2 model on the RPM and Gear spoofing attacks.	74
3.20	Per-message latency comparison against other state-of-the-art IDSs reported in the literature.	75
3.21	Per-message latency and measured power of the proposed model on the different platforms.	76
3.22	Resource utilisation on the PL for the proposed CAN IDS-ECU (XCZU7EV).	76

3.23	Number of parameters, input features and device used for the three IDS implementations.	77
3.24	Per-inference energy consumption and the chosen DPU configuration (clock frequency) for each IDS model.	77
3.25	Number of misclassifications for each model on all the four attacks. Each model was tested with 50000 messages for each attack.	78
3.26	Inference cost of the custom quantised models (CQMLP) inferred through the FINN library and normalised to the 4-bit model.	80
3.27	Confusion matrix capturing the classification results of the CQMLP. .	83
3.28	Accuracy metric comparison (%) of the CQMLP accelerator against the reported literature on the DoS, Fuzzing and RPM-Spoof attacks. .	84
3.29	Per-message latency comparison against other state-of-the-art IDSs reported in the literature.	85
3.30	Resource utilisation breakdown of the proposed CQMLP (XCZU7EV). .	86
4.1	Input features and training methods used by state-of-the-art IDSs & IPSs proposed in the research literature.	93
4.2	Inference accuracy metrics of CAE-CH pre- and post-quantisation (QCAE-CH) on all the attacks.	103
4.3	Inference accuracy metrics of CAE-SA pre- and post-quantisation (QCAE-SA) on all the attacks of the SA dataset.	103
4.4	Confusion matrix capturing the classification results of the QCAE-CH on the attack-free (benign), DoS, Fuzzing, and Spoofing (Gear, RPM) attacks. (Note the model is trained only on benign messages and does not see any attack message during the training phase.)	104
4.5	Confusion matrix capturing the classification results of the QCAE-SA on the attack-free (benign), Flooding, Fuzzing, and Malfunction attacks for the SA dataset. (Note the model is trained only on benign messages and does not see any attack message during the training phase.)	104
4.6	Accuracy metric comparison (%) of the QCAE-CH accelerator against the reported literature on the DoS and Fuzzing attacks.	106
4.7	Accuracy metric comparison (%) of the QCAE-CH accelerator against the reported literature on the RPM and Gear spoofing attacks. . . .	107

4.8	Accuracy metric comparison (%) of the QCAE-SA accelerator against the reported literature on the Flooding, Fuzzing and Malfunction attacks.	107
4.9	Inference latency comparison of QCAE-CH & QCAE-SA models against other state-of-the-art IDSs (supervised & unsupervised) reported in literature.	108
4.10	Performance scaling of the QCAE-CH & QCAE-SA model using different DPU configurations: impact on per message latency (ms), resource (%), & power consumption (W).	109
5.1	Accuracy metric comparison (%) of the QLSTM-IDS accelerator against the reported literature on the DoS and Fuzzing attacks for the car hacking dataset.	129
5.2	Accuracy metric comparison (%) of the QLSTM-IDS accelerator against the reported literature on the RPM and Gear attacks for the car hacking dataset.	130
5.3	Performance metric comparison of QLSTM-IDS against different models proposed for the Sonata set of the SA dataset.	134
5.4	Performance metric comparison of QLSTM-IDS against different models proposed for the Soul set of the SA dataset.	135
5.5	Performance metric comparison of QLSTM-IDS against different models proposed for the Spark set of the SA dataset.	136
5.6	Detection latency comparison of the proposed QLSTM-IDS against other state-of-the-art IDSs reported in the literature.	138
5.7	The resource consumption overheads of QLSTM-IDS and the CAN Controller on the PL region of the Zynq XCZU7EV SoC.	139
5.8	This table compares the proposed QLSTM-IDS with other QNN models from the research literature, evaluating them in terms of resource utilisation, energy consumption, and overall misclassification rates as reported on the car hacking dataset.	140
6.1	Comparison of detection accuracy (%) of the 4b-IDS solution on the car hacking (CH) and the survival analysis (SA) datasets.	153
6.2	IDS resource utilisation for the standard CAN controller & the proposed SecCAN controller.	155
6.3	Comparison of SecCAN's energy consumption per inference with other IDS approaches proposed in the literature.	155

List of Acronyms

ABS Anti-Lock Breaking System

ADAS Advanced Driving Assistance Systems

AUC Area Under Curve

AUTOSAR AUTomotive Open System ARchitecture

AXI Advanced eXtensible Interface

BRAM Block Random Access Memory

BRP Baud Rate Prescalar

CAE Convolutional Autoencoders

CNN Convolutional Neural Network

CSMA/CD Carrier sense multiple access with collision detection

CQMLP Custom Quantised Multi-Layer Perceptron

DoS Denial of Service

DPU Deep Learning Processing Unit

DSP Digital Signal Processor

ECU Electronic Control Unit

ECM Electronic Control Module

ESC Electronic Stability Control

EV Electronic Vehicles

FCA Fiat Chrysler automobiles

FF Flip Flops

FIFO First-In First-Out

FPGA Feild Programmable Gate Array

FPR False Positive Rate

FNR False Negative Rate

GAN Generative Adversarial Network

GBM Gradient Boosting Machine

GMM Gaussian Mixture Model

GRU Gated Recurrent Unit

GPS Global Positioning System

GPU Graphical Processing Unit

GTX Giga Texel Shader eXtreme

HLS High Level Synthesis

HVAC Heating-Ventilation-Air-Conditioning

IDS Intrusion Detection System

II Initiation Interval

IPS Intrusion Prevention System

IVI In-Vehicle Infotainment

LiDAR Light Detection and Ranging

LIN Local Interconnect Network

LTE Long-Term Evolution

LSTM Long Short Term Memory

LUT Loop Up Table

MAE Mean Absolute Error

MLP Multi Layer Perceptron

MCU Microcontroller Units

MOST Media Oriented Systems Transport

MSE Mean Squared Error

OBD On-board Diagnostic

OEM Original Equipment Manufacturers

ONNX Open Neural Network Exchange

OTA Over the air

PCIe Peripheral Component Interconnect Express

PL Programmable Logic

PMBus Power Management Bus

PS Processing System

PTQ Post Training Quantisation

PYNQ Python Productivity for Zynq

QAT Quantisation Aware Training

QONNX Quantised Open Neural Network Exchange

QoS Quality of Service

QNN Quantised Neural Networks

ReLU Rectified Linear Unit

ROC Receiver Operating Characteristic

RPM Rotation Per Minute

RTR Remote Transmission Request

RTL Register Transfer Level

SJW ynsynchronization Jump Width

SoC System on Chip

SVM Support Vector Machine

Tanh Hyperbolic Tangent

TCU Transmission Control Unit

TF TensorFlow

TPU Tensor Processing Unit

TSEG Time Segment

URAM Ultra Random Access Memory

UTP Untwisted Pair

VART VARTVitis-AI Runtime

V2V Vehicle-to-Vehicle

V2X Vehicle-to-Everything

Acknowledgements

To anyone reading this, I began working on this Ph.D. thesis with a lot of uncertainty, doubting my own abilities sometimes and unsure if I could live up to the expectations that come with producing good research. However, I was fortunate to have Dr. Shreejith Shanker as my guide. He never made me feel like I wasn't good enough. His feedback consistently pushed me forward and turned this four-year journey into a deeply rewarding and eye-opening learning experience. The many discussions I had with him shaped how I approach problems, encouraging me to seek practical and meaningful solutions. Words fall short of expressing my gratitude for the opportunity to be his first Ph.D. student. I will always wear that tag with pride.

I would also like to thank the Head of Department, Prof. Anil Kokaram, for his support and understanding during the period when I had to work from home due to certain constraints. I am sincerely grateful for his kindness and encouragement during that time. As the captain of our cricket team, he has taught me a lot, both on and off the field.

These four years have shown me that pursuing a Ph.D. can be a lonely journey, and it requires a solid support system to keep one going. I was lucky in this regard. During my time here, I met my partner, Dhruvi Jindal, who made sure I gave my best to this Ph.D. She challenged my limits and helped me reimagine what I was capable of. I'll admit I was stubborn (and still am), but she patiently worked through that. Without her, this journey would not have looked the same. She was my toughest critic, but also the one I needed the most. Thank you, Dhruvi, for all your support during this time.

I'd also like to thank my research peers who ensured there was never a shortage of humour in the lab. A special shoutout to Mohsen, who reminded us not to take life too seriously. Danilo and Arvind, thank you for being unapologetically yourselves, I'm sure you'll curse me for mentioning your names here, but that's okay. A heartfelt thank you to Chiara Paletta for being not only a fantastic mentor but also an even

more amazing person, Team Tiramisu for the win! To everyone else in the lab: thank you for your support and for being part of this chapter.

Special thanks to my dear friends, Yash and Shraddha. I'm truly grateful for the time you've shared with me. Thanks for being my family away from home in Dublin.

My friends back in India also deserve credit especially Shashwat Shrivastava & Sparsh Garg. I'm not entirely sure how they helped, but I know that our conversations made this ride more enjoyable for me. Sachin, Arhant, Ayush, and Yash, thank you for listening to my rants whenever I needed to vent. That made more of a difference than you might realise.

I am also deeply grateful to my parents, my sister, and my brother-in-law for always believing that I would give my best. Their support, encouragement, and love kept me grounded through it all. I was especially driven to succeed out of respect for the sacrifices they made for me throughout their lives. There were times when I was stuck on a problem for days, and during our calls, my sister would say, "You'll have it done by tomorrow", and she turned out to be right more often than I had imagined. I can't describe how important she was in this journey. My grandfather, I believe, would have been particularly proud of this achievement, I hope he is smiling wherever he is.

Lastly, if I've forgotten to mention someone and you're reading this, please know that you too deserve to be named here. Thank you.

Chapter 1

Introduction

1.1 Evolution of Vehicular networks

The evolution of automotive networks over the past fifty years reflects a transformative shift from simple mechanical control systems to highly sophisticated, interconnected digital platforms. This shift has fundamentally altered the design, functionality, and user experience of modern vehicles [1]. This advancement has been driven by the proliferation of advanced electronic control systems, to support the increased demands for vehicular safety, enhanced user experience, and environmental considerations [2, 3]. In the early days of automotive systems, control mechanisms were primarily mechanical and hydraulic, with limited or no electronic components for monitoring or optimising performance. With advancements in technology, electronic assistance systems were increasingly adopted to improve vehicular emissions and passenger safety, in addition to new capabilities such as entertainment [4]. This trend has since continued with new and advanced technologies, such as driver assistance systems (ADAS) and high-bandwidth sensors, elevating both the complexity and the features supported by modern cars [5]. As a result, vehicles today are intelligent, connected machines capable of real-time adaptive control, with myriad sensors that can communicate over internal networks and share data with external infrastructures.

1.1.1 Early Developments and the Introduction of ECUs

The 1980s marked a pivotal milestone in the evolution of automotive electronics by introducing electronic control units (ECU) – a software-based real-time control system executing on microcontrollers. These microcontroller-based systems allowed for precise control of critical functions such as fuel injection and ignition timing,

leading to improved fuel efficiency, reduced emissions, and enhanced vehicle reliability. With minimal sensors, they relied on direct point-to-point links between sensors, actuators and the ECU. This success led to a wider appetite for electronic-controlled functionality, with ECU operating independently, with no efficient means (or requirement) to share data across systems, resulting in functional silos [6]. This siloed architecture limited the overall efficiency and coordination of vehicle subsystems, prompting the need for more integrated solutions.

1.1.2 Emergence of Intra-Vehicular Networks: The CAN Protocol

As the adoption of electronic control units expanded, the limitations of point-to-point connectivity and siloed functions became increasingly evident, resulting in greater vehicle weight and inefficiencies. To address these challenges, Bosch introduced a standardised in-vehicle communication framework in 1986: the Controller Area Network protocol [7]. This protocol provided a reliable and robust communication mechanism for ECUs, sensors, and actuators, facilitating seamless integration via twisted-pair cabling in contrast to the complex and labor-intensive point-to-point wiring. Recognising the inefficiencies of isolated ECUs, Bosch designed the controller area network (CAN) protocol to enable reliable communication over a shared bus system [7]. By eliminating the need for intricate star topologies that required direct connections between individual ECU pairs, CAN significantly reduced production costs, minimised vehicle weight, and improved communication efficiency. Moreover, the protocol incorporated an arbitration mechanism at the protocol level, allowing mixed-critical ECUs to share the same network while optimising wiring costs and overall system weight. By the 1990s, CAN had gained widespread acceptance and was adopted as the standard in-vehicle network for critical and non-critical ECU functions. The availability of an efficient shared-bus protocol spurred further advancements in electronic-controlled functionalities and led to the domain-based segregation of vehicular systems. This architectural approach categorised functions into domains such as powertrain control, body electronics, and infotainment, ensuring the isolation of critical tasks from non-critical systems to enhance safety and security. However, while CAN was well-suited for critical automotive communications, it proved too complex and costly for basic body domain functions and too slow for emerging multimedia applications. This necessitated the introduction of specialised protocols tailored to specific use cases. For instance, the Media Oriented Systems

Transport (MOST) protocol was developed to support high-speed audio and video data transmission, while the Local Interconnect Network (LIN) protocol was designed for lower-speed body functions such as window control and climate regulation [8, 9]. Despite the emergence of alternative protocols, CAN remained the backbone of in-vehicle networks, serving as the primary communication infrastructure for critical ECU interactions due to its robustness, reliability, and cost-effectiveness [10].

1.1.3 Integration of Infotainment and Connectivity Systems

The late 1990s and early 2000s marked the integration of advanced infotainment systems, GPS-based navigation, satellite radio, and in-vehicle telephony, significantly expanding the scope of vehicular networks. As cellular networks evolved to support higher data transfer rates, the advent of 4G LTE connectivity in the 2010s enabled vehicles to establish communication with external servers and other vehicles. This period saw the introduction of Vehicle-to-Vehicle (V2V) and Vehicle-to-Everything (V2X) communication paradigms, facilitating advancements in intelligent traffic management and pedestrian safety [11, 12]. Simultaneously, the development of Advanced Driver Assistance Systems (ADAS), incorporating cameras, radar, and LiDAR technologies, enabled functionalities such as adaptive cruise control, lane departure warnings, and automated emergency braking [13]. The data-intensive nature of these emerging systems necessitated the adoption of Ethernet-based architectures, capable of supporting low-latency, high-bandwidth communication to meet the increasing demands of modern automotive networks [14].

Today, modern vehicles represent a convergence of mechanical, electronic, and software systems. They function as intelligent, connected machines capable of real-time data exchange, complex decision-making, and autonomous operation. This transformation highlights the rapid evolution of automotive networks from isolated, mechanically-driven control systems to highly integrated, software-defined platforms. As this evolution continues, future automotive networking is expected to be shaped by advancements in artificial intelligence, 5G connectivity, and edge computing—pushing the boundaries of vehicle safety, efficiency, and user-centric innovation. Despite these advancements, modern automotive architectures still adopt a modular design, where computations are organised into functional domains and handled by ECUs communicating over partitioned in-vehicle networks. These domains typically separate critical and non-critical functionalities; however, regulatory and maintenance demands—such

as the need for On-Board Diagnostics (OBD) access—have increasingly eroded this separation, reducing network isolation [15].

1.1.4 Challenges in Security and Cybersecurity Vulnerabilities

While advancements in automotive networking continue to deliver substantial benefits in safety, efficiency, and user experience, security considerations were given lower priority. Primarily, this stemmed from the self-contained nature of vehicular electronic functions with siloed critical functions. Network protocols like CAN and FlexRay had limited intrinsic security features, as they were developed for siloed automotive networks. The shift toward connected cars, now numbering hundreds of millions worldwide, has introduced significant security challenges. Modern vehicles integrate around 100 ECUs managing critical functions and autonomous assistive systems, which rely on increasing levels of connectivity. To enable capabilities such as remote diagnostics and over-the-air (OTA) updates, traditionally siloed networks must be bridged, inadvertently creating new attack surfaces that adversaries can exploit remotely through wireless networks and OTA mechanisms [16–18]. Despite their convenience, these OTA interfaces can introduce potential attack vectors. Malicious actors can gain unauthorised access to critical systems, as demonstrated in high-profile research studies, highlighting the need for comprehensive cybersecurity strategies. The lack of universal security standards and the cost-sensitive nature of the automotive industry have slowed the adoption of robust protection mechanisms. Consequently, ensuring the security of automotive networks remains a pressing challenge, particularly as vehicles become increasingly autonomous and connected [19].

In summary, the evolution of automotive networks has been a journey from basic control systems to complex, interconnected platforms. Each phase has brought new capabilities and challenges, shaping the trajectory of modern vehicular technology. Addressing security concerns while balancing cost, performance, and innovation will be critical for the future of intelligent transportation systems.

1.2 Motivation: Efficient & Scalable IDS for Automotive E/E

With increasing automation and connectivity, vehicular systems will require robust security systems that can be adapted throughout the vehicle’s lifetime, either through

over-the-air updates or through authorised channels (dealers, workshops). In modern vehicles, interfaces like Bluetooth, Wi-Fi, and telematics present potential entry points for attackers to access otherwise siloed vehicular systems and perform remote attacks. These interfaces can be exploited to control critical functions, including acceleration and braking, via intra-vehicular networks [16].

As discussed previously, legacy automotive protocols such as CAN lack fundamental security features. These vulnerabilities include the absence of authentication and reliance on broadcast-based communication, making it possible for attackers to eavesdrop on network traffic or execute injection attacks [20]. Such security breaches can compromise critical vehicular systems, resulting in privacy violations for vehicle owners or even endangering passenger safety.

In response to the growing prevalence of cyberattacks, regulatory bodies have begun instituting cybersecurity guidelines and mandates. For instance, the United Nations Economic Commission for Europe (UNECE) introduced the WP.29 regulation, which requires comprehensive cybersecurity risk management for vehicles [21]. Similarly, the U.S. National Highway Traffic Safety Administration (NHTSA) has issued its automotive cybersecurity guidelines [22].

Intrusion Detection Systems (IDS) represent a pivotal first line of defence for modern vehicular security. By monitoring network traffic patterns on the network, they can be trained to detect unauthorized access and attacks in real-time. When IDS systems identify anomalous behaviour, they can trigger defence mechanisms to safeguard both passengers and the vehicle. Specifically, IDSs have shown to be extremely successful in detecting and mitigating active injection and spoofing attacks, preventing malicious messages from compromising critical systems [18, 23]. As vehicles evolve toward higher levels of autonomy, IDSs ensure sensor data integrity, preventing manipulated inputs that could lead to unsafe behaviour/ operating conditions [24].

Advanced IDSs can also be designed to detect zero-day attacks – those exploiting previously unknown vulnerabilities – which pose significant risks to vehicle owners and manufacturers alike [25]. Upon identifying zero-day threats, IDSs can initiate defence responses, and OTA updates can be deployed to patch affected systems. The myriad sensors and actuators in modern vehicles continuously generate data which is exchanged over in-vehicle networks to support safety-critical functions and performance optimisations. Such data also facilitates real-time decision-making to enhance driver assistance systems and autonomous capabilities through coordination between vehicle components and external infrastructure in connected environments.

However, the complexity and intertwined nature of these data exchanges create security challenges, as malicious actors may attempt to intercept, manipulate, or exploit vulnerabilities within the system. Without robust cybersecurity measures, such attacks could compromise vehicle functionality, jeopardize passenger safety, and disrupt transportation networks. As regulatory standards for vehicular cybersecurity evolve, IDS is positioned to become a fundamental component of a comprehensive defence strategy [26].

Research in vehicular cybersecurity has primarily focused on developing IDS solutions which analyse intra-vehicular network traffic to detect potential anomalies. While specifics of real-world deployments remain largely undisclosed by manufacturers, IDS research in the research domain explores a range of threat mitigation strategies – from alerting drivers via warning lights or error messages to transitioning vehicles into a minimal "safe operating mode" to contain threats [27]. Given their potential for real-time attack detection, IDSs continue to be a major area of exploration in securing future connected and autonomous vehicles. The following subsections highlight the major areas of interest while developing IDS solutions for CAN.

1.2.1 Computational Complexity

To perform real-time analysis of network traffic, different algorithms, and implementation strategies have been explored in the literature. Early IDS research explored rule-based and signature-based detection methods. Rule-based systems defined operational parameters, such as the frequency of ECU transmissions, flagging deviations as potential threats. Signature-based IDSs matched current network activity to known attack patterns. However, these systems struggled with scalability, high false-positive (leading to normal messages being flagged as attacks) and false-negative rates, and memory overheads [28, 29]. In contrast, machine learning (ML)-based IDSs have been shown to offer improved accuracy, adaptability to novel threats, and generalisability by learning normal traffic patterns and identifying deviations more effectively than rule-based approaches [30]. The full-precision models however proposed in the literature, incur significant computational complexity which makes their deployment on real-vehicles non-trivial. The ML-based IDSs need to satisfy multiple objectives before they are suitable for deployment namely:

- **Classification Accuracy:** High attack classification accuracy is essential due to the volume of CAN traffic. With up to 10,000 messages per second on a fully utilized bus, even 99% prediction accuracy results in 100 misclassified messages

per second. IDSs hence must achieve at least 99.9% prediction accuracy or more to ensure reliable communication so that normal messages are not dropped from the network.

- **Detection Latency:** The IDS needs to have the ability to process 10,000 messages per second. This throughput translates and restricts the per-message detection latency of the IDS to be not greater than 100 micro seconds and in an ideal scenario even faster to allow for preventative measures to happen.
- **Energy Efficiency:** Energy limitations in vehicular environments demand energy-optimised IDS designs. Resource-intensive ML models (known to be highly accurate) must be adapted for low-power environments without sacrificing performance.

1.2.2 Generalisability and Evolving attack vectors

The increasing connectivity in modern vehicles has rendered CAN to be vulnerable to a growing range of cybersecurity threats. As attackers develop more sophisticated and varied methods to exploit vehicular networks (with attacks like DoS, fuzzing, specific component spoofing or malfunction vectors), IDSs for CAN must evolve to remain effective. One of the critical challenges in this context is the need for IDSs to generalise, that is, to learn and adapt to patterns of normal communication while also being capable of detecting novel and previously unseen attack vectors. This requirement arises from the fact that attack surfaces in automotive systems are not static. As new ECUs, sensors, and telematics units are integrated, the potential for vulnerabilities expands, and the nature of attacks continues to change. Hard-coded rules or signatures become insufficient, as they cannot accommodate unknown or evolving threats. Therefore, a robust IDS must exhibit strong generalisability over the dynamic communication patterns present in CAN traffic.

However, modelling normal communication between ECUs presents a significant challenge. Unlike traditional computer networks, CAN traffic is characterised by a combination of periodic, event-driven, and often proprietary message formats. This complexity makes it difficult to build a complete and reliable statistical model of normal behaviour using conventional methods. Furthermore, legitimate CAN messages can vary based on vehicle context, driving conditions, and system configurations, making the boundary between benign and malicious traffic inherently fuzzy. As CAN-based vehicles continue to operate in increasingly complex and unpredictable

environments, the ability to generalise and adapt to new attack patterns should be the cornerstone for the next generation of intelligent and resilient in-vehicle IDSs.

1.2.3 Integration Challenges

While numerous research studies have highlighted the advantages of ML-based IDSs, their deployment within vehicular electric/electronic (E/E) systems presents significant challenges. The heterogeneous nature of vehicular networks (with multiple protocols and domains) combined with the limited computational capabilities and power budgets makes embedding IDS functionality challenging. Functional domains differ in terms of messaging frequency, strategy, bandwidth, and latency; additionally, they may employ traditional event-based protocols such as CAN and LIN or higher-bandwidth time-triggered protocols such as FlexRay and Automotive Ethernet, with distinct characteristics in terms of network access, security and bandwidth. This diversity complicates the seamless integration of IDS solutions, requiring adaptable architectures that can operate effectively across varying network topologies while maintaining efficiency and real-time performance. Furthermore, the computational complexity of ML models makes them difficult to deploy on automotive-grade microcontrollers with limited computational capability and power budget. High-performance architectures like GPUs are capable of real-time classification; however, their higher power consumption and complex interfacing (PCIe) limits their suitability for in-vehicle use. Sharing critical ECU resources with IDS functionality can introduce additional latency, affecting critical applications [18, 31, 32].

Addressing these issues requires specialised architectures that can minimise the impact on critical ECU functions. Loosely coupled accelerators for offloading such tasks can provide the required computational power while incurring higher energy overheads and software overheads on the ECU. With a reconfigurable platform (such as an FPGA), the accelerator could be optimised and tightly coupled to the ECU to minimise energy and software overheads. While embedding novel capabilities such as fault tolerance [33]. Hybrid FPGA-based ECU architectures have been shown to offer the ability to provide task consolidation with clear partitioning of critical and non-critical functions on a single die [34]. Given these advantages, an FPGA-based approach to develop energy-efficient, real-time IDS solutions can be tailored for in-vehicle networks. This approach not only aligns with the performance and energy constraints of automotive systems but also supports flexibility in adapting to evolving

attack patterns. The reconfigurability of FPGAs facilitates rapid prototyping and deployment of new detection models as threats emerge.

1.3 Research Questions

The motivations to meet the IDS constraints presented in the previous subsection shape the research questions that drive this thesis, focusing on advancing IDS solutions for enhanced vehicular security and adaptability which are present next:

1. **Optimising ML-based IDS for Automotive Networks:** Can ML-based IDS models be improved by leveraging reduced precision and sparsity to minimise computational overhead while maintaining high detection accuracy? Subsequently, can minimal overhead networks be derived that offer the best trade-off between precision, architecture simplicity, and low-latency detection?
2. **Scalability and Generalisation:** Can ML-IDS models effectively classify multiple attack types while adapting to unknown threats? Additionally, can the IDS achieve generalisation across different message sets and vehicles?
3. **Time-series analysis:** Can the time-series nature of vehicular messages be exploited using LSTMs to remove the bottleneck on windowing and block-based detection? If so, can such models be compressed through quantisation for in-vehicle deployment?
4. **Integration and Deployment:** Can detection latency and software overheads on the ECU be alleviated by embedding the IDS into the network controller, paving the way for smart-CAN and smart-Ethernet network interfaces?

1.4 Contributions

In this thesis, a comprehensive exploration of novel IDS solutions is presented tailored towards lightweight field programmable gate array (FPGA) devices, designed for multiple types of IDS deployment architectures (consolidated/distributed) in vehicular networks. The proposed approach emphasises isolating the IDS function from the main ECU tasks while maintaining full software control over its execution. By leveraging lightweight ML models and evaluating them on multiple publicly available

automotive attack datasets, a robust and efficient methodology is demonstrated for their development, validation, and deployment while satisfying deployment constraints.

Two approaches are evaluated for deployment on AMD’s Zynq Ultrascale+ FPGAs. First, a software-driven IDS strategy is explored using AMD’s Deep-learning Processing Unit (DPU) integrated as a coupled accelerator to offload IDS capability from the ECU. Here, the benefit of exploiting quantisation and sparsity of the ML models is shown and offloading optimisations are investigated to minimise ECU overheads. Subsequently, custom-designed accelerators are investigated for lower precision and tighter integration with the ECU and network controller. Here, the networks are further refined with quantisation to achieve the highest model compression, and implementation optimisations are explored to minimise latency and energy consumption. The strategies offer flexible integration pathways and tradeoffs for performance, energy efficiency, and upgradability.

The key contributions of this thesis are outlined as follows:

- Four critical deployment constraints for intrusion detection systems in real-world vehicles have been identified namely: *classification accuracy*, *real-time latency*, *energy efficiency*, and *resource utilisation*. Unlike many studies in current literature, this thesis directly addresses these constraints.
- The presented results are among the first to demonstrate that lightweight deep learning models optimised through network quantisation strike the optimal balance between accuracy, latency, resource utilisation, and energy consumption, paving the way for practical vehicular IDS deployment.
- A custom low-precision feed-forward Quantised Multi-Layer Perceptron (QMLP) and Quantised Convolutional Neural Network (QCNN) models are developed, which employ block-based message processing for CAN traffic along with the integration architecture within a model ECU. The best model achieves state-of-the-art classification performance across diverse attack vectors ($> 99.9\%$ classification accuracy, 0.1 ms processing latency ($2.2 \times$ reduction against SOTA) and $\approx 0.23\text{ mJ}$ per inference energy consumption ($3.9 \times$ reduction against SOTA)).
- To detect zero-day attacks, an unsupervised learning-based quantised convolutional autoencoder model is presented that offers high accuracy ($> 99.5\%$) in identifying unseen threats within vehicular networks and its integration into

a vehicular ECU (real-time attack detection and ≈ 2.1 mJ per inference ($2\times$ reduction against SOTA)).

- Integration of sequential ML models is presented, specifically Long Short-Term Memory Networks (LSTMs), to support per-message multiclass attack classification, ensuring real-time compliance while satisfying the deployment constraints ($> 99.9\%$ per-message classification accuracy, 0.25 ms processing latency and ≈ 0.8 mJ per inference energy consumption).
- A comprehensive deployment architecture for all the models is presented on hybrid FPGA-based ECUs compliant with AUTOSAR standards, demonstrating the capacity to detect multiple attacks across the utilised datasets while maintaining fully isolated IDS execution.
- Secure network controllers with embedded IDSs are presented that enable proactive attack detection before messages are transmitted to ECUs, facilitating early intervention ($> 99\%$ binary attack classification accuracy, 36.5 μ s processing latency and ≈ 73.7 μ J per inference energy consumption).

1.5 Thesis Organisation

The remainder of the thesis is organised as follows. Chapter 2 provides background information on automotive systems, ECUs, the CAN protocol, IDS approaches, and earlier examples of how machine learning models are deployed on FPGAs today. Chapter 3 presents the initial study that investigated model compression using quantisation to improve the latency and energy overheads of IDS solutions. DPU-based offload case studies have been presented and custom quantised dataflow accelerators as effective IDS solutions with state-of-the-art accuracy, inference latency, and energy consumption when tested across different attack vectors. In chapter 4, an unsupervised learning-based quantised convolutional autoencoder model is presented as a generalised anomaly detector and its use to perform zero-day attack detection. The training, validation, and implementation strategies for the zero-day model are discussed in this chapter. In chapter 5, recurrent neural networks have been investigated as they capture the time-series nature of network messages and tool flows have been developed that can map them into dataflow accelerators. The key benefit of the quantised LSTM model is its ability to flag individual messages as opposed to the block and window-based methods that analyse and flag blocks/windows of

CAN messages as infected. Mapping optimisations have also been developed for the quantised LSTM model to reduce the inference latency of the complex model. In chapter 6, a network-offload capability is investigated by integrating a quantised neural network model within the datapath of an otherwise standard CAN controller. This approach is completely transparent to the ECU and presents zero software overheads to the function, making them ideally suited for critical and constrained ECU subsystems. Additionally, by carefully overlapping detection and network reception, the approach hides the IDS latency and can be easily augmented to enable/activate early preventative measures directly at the network interface. Chapter 7 discusses the summary and future works of the thesis and the thesis is concluded in this chapter.

1.6 Publications & Open-source contributions.

The following is the list of publications produced during the thesis duration.

- S. Khandelwal, E. Wadhwa and S. Shreejith, "Deep Learning-based Embedded Intrusion Detection System for Automotive CAN," IEEE International Conference on Application-specific Systems, Architectures and Processors (**ASAP**), Gothenburg, Sweden, 2022, pp. 88-92
- S. Khandelwal and S. Shreejith, "A Lightweight Multi-Attack CAN Intrusion Detection System on Hybrid FPGAs," International Conference on Field-Programmable Logic and Applications (**FPL**), Belfast, United Kingdom, 2022, pp. 425-429
- S. Khandelwal and S. Shreejith, "A Lightweight FPGA-based IDS-ECU Architecture for Automotive CAN," International Conference on Field-Programmable Technology (**ICFPT**), Hong Kong, 2022, pp. 1-9
- S. Khandelwal and S. Shreejith, "Real-Time Zero-Day Intrusion Detection System for Automotive Controller Area Network on FPGAs," IEEE International Conference on Application-specific Systems, Architectures and Processors (**ASAP**), Porto, Portugal, 2023, pp. 139-146
- S. Khandelwal and S. Shreejith, "Exploring Highly Quantised Neural Networks for Intrusion Detection in Automotive CAN," International Conference on Field-Programmable Logic and Applications (**FPL**), Gothenburg, Sweden, 2023, pp. 235-241

- S. Khandelwal, A. Walsh and S. Shreejith, "Quantised Neural Network Accelerators for Low-Power IDS in Automotive Networks," Design, Automation & Test in Europe Conference & Exhibition (**DATE**), Antwerp, Belgium, 2023, pp. 1-2
- S. Khandelwal, S. Shreejith, "SecCAN: An Extended CAN Controller with Embedded Intrusion Detection", **IEEE ESL: Embedded System Letters**, May 2025
- S. Khandelwal, J. Konig, T. Pressuer, M. Blott, S. Shreejith, "FINN-GL: Generalized Mixed-Precision Extensions for FPGA-Accelerated LSTMs", International Conference on Field-Programmable Logic and Applications (**FPL**), Leiden, The Netherlands, 2025

Papers under review :

- S. Khandelwal, S. Shreejith, "Deep Defence on Wheels: A dual-IDS framework for Comprehensive In-Vehicle Network Security", **ACM TECS - Transactions on Embedded Computing Systems**

The following is the list of open source contributions proposed in the thesis.

- **Chapter-3:** CQMLP
- **Chapter-4:** QCAE
- **Chapter-5:** QSLTM-ONNX Frontend; QLSTM-IDS
- **Chapter-6:** SecCAN

Chapter 2

Background

I predict that this will likely be the first multitrillion-dollar robotics industry.

Jensen Huang on autonomous
vehicle industry - CES 2025

This chapter builds upon the foundational concepts introduced in the previous chapter by providing a detailed discussion of the topics explored in this thesis. An overview of automotive ECUs is presented leading into their interconnected nature and the different types of automotive network protocols, highlighting their evolution and functional significance. Next, the security challenges associated with these networks are discussed, emphasising vulnerabilities and attack surfaces. Following this, the range of IDS solutions proposed in the research literature are discussed, identifying key limitations primarily arising from the deployment constraints of resource-constrained vehicular environments. Finally, the proposed deployment platform is introduced for the IDS solutions presented in this thesis, detailing its prior use in automotive applications and how it effectively supports the implementation of optimised machine learning algorithms that can be tailored for real-time vehicular security.

2.1 Automotive Electronic Control Units

An ECU also referred to as an electronic control module (ECM), is typically an embedded computing device, predominantly employing a microcontroller at its heart, that executes software routines to manage one or more electrical subsystems within

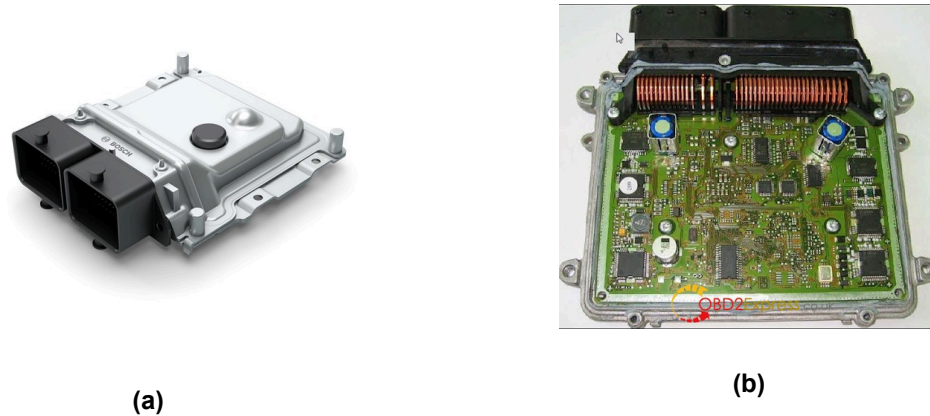


Fig. 2.1 The figure depicts the internal and external structure of a typical ECU.

the vehicle. Multiple such embedded devices are networked using an automotive network, such as CAN or LIN, to achieve safety-critical and control functions in modern cars.

Modern vehicles rely on a network of ECUs, each dedicated to managing specific functionalities such as braking, door locking, or airbag deployment. ECUs typically execute software tasks that implement parts of or entire functional components and operate independently. External interfaces are used to power and interconnect the ECU via vehicle-wide communication protocols such as CAN. To implement functions, ECUs rely on inputs from a wide array of sensors distributed throughout the vehicle – and in some cases, external sources such as key fobs. The software tasks periodically probe the input values from the sensors and generate activation commands when an ‘event’ is detected from the combination of sensor values. The tasks are run with tightly controlled timing to ensure that sensing and actuation occur at deterministic times. This applies to simple functions such as door locks that actuate the mechanical locking action of a door, to complex safety-critical functions such as deploying airbags and engaging emergency brakes when a possible collision is detected. ECUs thus form the core of a vehicle’s cyber-physical system, bridging the gap between environmental sensing and responsive mechanical/electronic actuation. Figure 2.1 illustrates the internal structure of an ECU, including its electronic components and interfaces.

Over the years, the role of ECUs has evolved significantly alongside advancements in automotive technology. Initially, ECUs were primarily tasked with basic engine management functions, such as controlling fuel injection and ignition timing. However, their functionality has greatly expanded to include a wide array of vehicle systems.

Vehicle Type	Number of ECUs
Economy Car	30–50
Mid-Range Sedan	70–100
Luxury Vehicle	100–150
Electric Vehicle (EV)	120–150
High-End Sports Car	100–140

Table 2.1 The table displays the various vehicle types along with the corresponding number of ECUs in each vehicle.

Vehicle Type	Primary Functions
Economy Car	Basic engine management, airbag deployment, anti-lock braking system (ABS), and climate control.
Mid-range sedan	Advanced engine management, transmission control, infotainment system, safety systems (airbags, ABS, ESC), and basic driver assistance.
Luxury Vehicle	Sophisticated ADAS (lane-keeping, adaptive cruise control), high-end infotainment, climate control, and advanced safety systems.
Electric Vehicle (EV)	Battery management, regenerative braking, electric motor control, ADAS, and connected vehicle technologies.
High-End Sports Car	High-performance engine management, advanced transmission control, ADAS, and driver-focused infotainment systems.

Table 2.2 The table outlines the different vehicle types and the main functions performed by the ECUs in each vehicle.

Today, a modern vehicle may feature up to 150 ECUs, and the embedded software is becoming increasingly complex and expansive. Table 2.1 & 2.2 show the number of ECUs and the functions they serve across different categories of vehicles. These advanced ECUs now handle critical tasks such as transmission management, and the operation of safety systems like anti-lock brakes and airbags, and play an integral role in advanced driver assistance systems (ADAS). ADAS relies on automated technologies, including sensors and cameras, to detect obstacles or driver errors and respond appropriately to enhance safety and performance [35]. Examples of different types of ECUs in vehicles and their primary functions are as below:

- **Engine Control Unit (ECU):** Monitors and adjusts engine parameters such as fuel injection, ignition timing, and emissions.

- **Transmission Control Unit (TCU):** Ensures smooth gear transitions.
- **Anti-lock Braking System (ABS):** Monitors wheel speed sensors and modulates brake pressure to prevent wheel lock-up during braking.
- **Electronic Stability Control (ESC):** Prevents skidding by automatically activating the brakes to help steer the vehicle correctly.
- **Airbag System:** Comprises airbags and sensors, deploying inflatable cushions during collisions to protect occupants.
- **Climate Control System:** Regulates heating, ventilation, and air conditioning (HVAC) functions.
- **Lighting and Wiper Systems:** Controls automatic headlight activation, adaptive lighting, and intermittent wiper operations.

As more and more electronic-controlled functions are added to vehicles, the number of ECUs and the complexity of the processing components within them have been on the rise. Many critical applications rely on multi-core CPU architectures and/or specialised computing offered by FPGAs and GPUs to meet the real-time requirements of these functions. A key challenge faced by original equipment manufacturers (OEMs) and Tier-1 suppliers is to find ways to control the number and complexity of ECUs [36].

Each ECU is a specialised embedded system composed of a microcontroller, memory, and interface logic, designed to perform specific tasks such as sensing, processing, and actuation. In modern vehicles, ECUs are organised into functional domains—like powertrain, safety, and infotainment—based on their criticality and role. While domains help isolate functions, ECUs often need to coordinate with others across domains. This communication occurs over shared in-vehicle networks such as the CAN bus, where messages are exchanged either as broadcasts or directed requests. Although effective, this interconnected setup introduces challenges in real-time coordination and security, particularly given the open nature of these communication channels.

2.2 In-vehicle networks

Modern automotive systems are organised into functional domains, such as powertrain, chassis, safety, and infotainment, each comprising multiple ECUs that must

communicate reliably. To support this inter-domain communication, vehicles employ several in-vehicle networks, with CAN being the most widely used due to its robustness, real-time performance, and low cost. The following subsections explore these networks in more detail, highlighting their role in enabling coordinated and efficient ECU operations.

2.2.1 Controller Area Network

In-vehicle networks enable distributed ECUs to exchange control and data messages to achieve the global functions of the vehicle. Multiple protocols are used in vehicular systems to cater to different functions based on their criticality and to optimise the cost of E/E systems. CAN [7] and CAN-FD [37] continue to be the most widely used protocols today due to their lower cost, flexibility, and robustness. CAN bus is implemented physically using the twisted pair cable as shown in figure 2.2. It is also used inside vehicles, built environments (e.g., controlling elevators in buildings, building energy management systems), railway applications, medical devices, and aircrafts [38]. Figure 2.3 illustrates the bit-field definition of a CAN data frame, with each segment providing some function in the network operation. Each bit-field within the CAN frame serves a specific purpose: start of frame (SOF) bit indicating the start of a new message, remote transmission request (RTR) bit to request information from another ECU, acknowledge (ACK) field to indicate errors in frame and the cyclic redundancy check (CRC) field carrying a CRC value.

Four types of CAN frames exist depending on their functionality. These are as follows:

- **Data frame:** This is the only frame among the four used for actual data transmission between nodes. The length of the payload in a CAN data frame varies from 0 to 8 bytes. This frame comes in two formats:
 - **Base frame format:** The identifier length is 11-bits.
 - **Extended frame format:** The identifier length is 29-bits ($11\{\text{base identifier}\} + 18\{\text{identifier extension}\}$).

The distinction between the CAN base frame format and CAN extended frame format is made by using the IDE bit, which is transmitted as dominant ('0') in the case of an 11-bit frame, and transmitted as recessive ('1') in the case of a 29-bit frame.

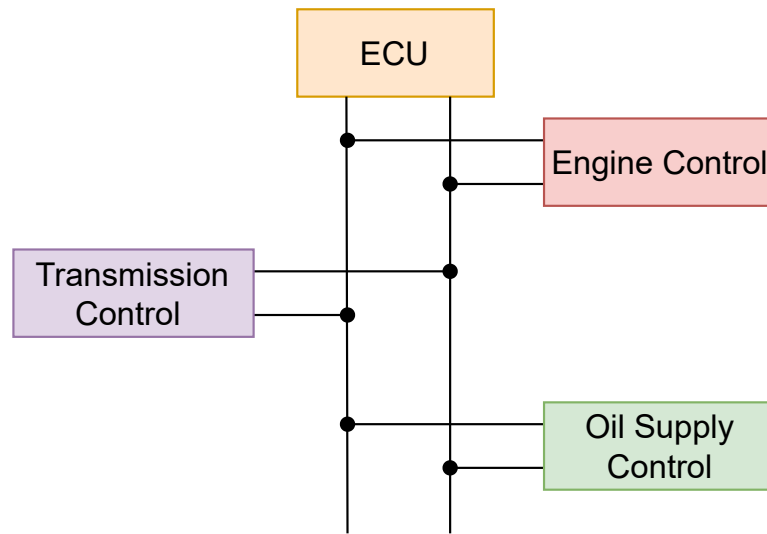


Fig. 2.2 The figure illustrates the shared bus-based topology of CAN in in-vehicle networks.

- **Remote frame:** It is also possible for a node to request information from another node in the network. This is achieved with the help of a remote frame. There are two differences between data and remote frames as follows:
 - The RTR bit is transmitted as a dominant bit in a data frame and as a recessive bit in the remote frame.
 - The remote frame does not have a payload field as indicated by its DLC.
- **Error frame:** There are five types of error defined in the Controller Area Network namely bit error, stuff error, CRC error, form error, and ACK error. Whenever any of these errors are detected, the error frame is sent to the bus for all the nodes to drop the error message. Figure 2.4 shows the error frame format for CAN. Each node in the bus has a transmit error count (TEC) and a receive error count (REC) associated with it. The state diagram 2.5 shows the node behaviour for different ranges of these values [39].
- **Overload frame:** This frame is used when a delay needs to be injected between a data frame and a remote frame.

The CAN ID is a unique identifier assigned to each message and by extension, defines its priority for transmission on the shared bus via the Carrier Sense Multiple



Fig. 2.3 Frame format of an extended frame CAN message. In the case of a standard CAN message, the ID field contains the 11-bit identifier.

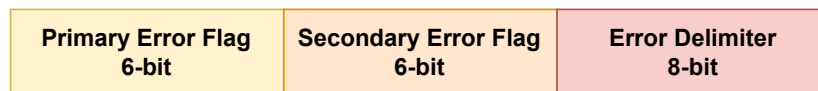


Fig. 2.4 Frame format of an error frame CAN message.

Access with the Collision Avoidance and Arbitration Priority (CSMA/CA-AP) access/arbitration mechanism. It is also used as an identifier to help ECUs determine whether they should process an incoming message on the bus or not by using filtering techniques. This is necessary since the CAN traffic is broadcast in nature. All ECUs receive all CAN packets and must decide whether they are intended for them. The maximum bitrate is 1 Mbps in the classical CAN Bus with a payload of up to 8 bytes, with the high-speed CAN Bus bitrate varying from 125 kbps to 1 Mbps and the low-speed CAN Bus bitrate from 5 kbps to 125 kbps; in the CAN Flexible Data (FD) rate, the bitrate is up to 8 Mbps with a payload size of up to 64 bytes per message. CAN XL is the third generation, providing up to 2,048 bytes of data payload and a bitrate of up to 10 Mbps, it is expected to be used with Internet Protocol-based services [40]. While critical sensors are connected to the high-speed CAN Bus, less-critical sensors can be connected to a low-speed CAN Bus.

CAN though is inherently insecure: there is no built-in mechanism in the network to authenticate the transmitter, receiver, or the message content itself [41]. The nodes in CAN don't know where the message associated for them is coming from. For example, a data frame carrying information about engine stoppage can come from the infotainment ECU and still be considered valid. This makes CAN vulnerable to simple and efficient attacks like message sniffing, fuzzing, spoofing, replay attacks, and Denial of Service (DoS) attacks [42–45]. Since there is a broadcast-based message transmission mechanism in place; the attacker can send harmful frames to any node on the network once it is in control of any node. This mechanism also can lead to passive sniffing attacks where an attacker could silently capture all the frames (which is critical information) being transmitted on the bus and the use this information to carry out replay attacks or more targetted attacks like spoofing which are harder to detect [46]. The above sniffing attacks are relatively straightforward as

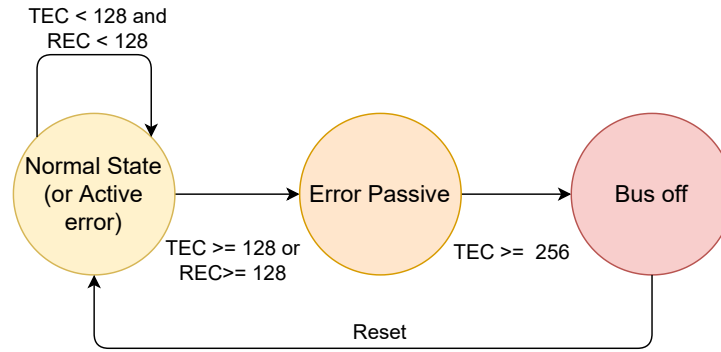


Fig. 2.5 State diagram for error handling in CAN.

CAN communication is unencrypted over the bus which makes it straightforward for attackers to decipher critical network information. These attacks rely on access to the CAN network; hence onset of such attacks indicates compromised ECUs on the critical network and thus detecting such intrusions is of key interest.

2.2.2 Automotive Ethernet

Automotive Ethernet is a specialised version of Ethernet designed to meet the unique requirements of automotive applications. Unlike traditional Ethernet, Automotive Ethernet features a physical layer specifically adapted to handle the stringent demands of the automotive environment, including high levels of electromagnetic compatibility (EMC) and immunity to interference. This is achieved through the use of advanced PHY transceivers that minimize the cost of cabling while maintaining reliability under the challenging conditions typical in vehicles, such as temperature extremes, vibration, and electrical noise [47]. Figure 2.6 shows the topology of the standard automotive ethernet network.

Sensor	Bit Rate per Sensor
Camera	500–3500 Mbps
Lidar	20–100 Mbps
Radar	0.1–5 Mbps
Ultrasound	0.01 Mbps

Table 2.3 The table provides the estimated bandwidth requirements for various sensors commonly used in in-vehicle networks.

Automotive Ethernet gained traction through Broadcom’s BroadR-Reach initiative, which laid the foundation for high-speed, cost-effective Ethernet communication

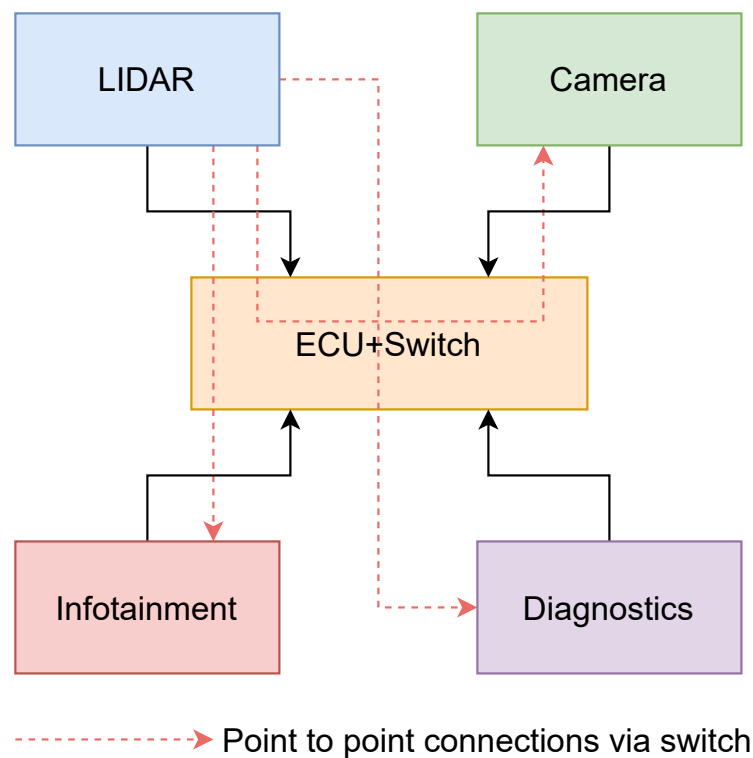


Fig. 2.6 The figure illustrates the star topology of Automotive Ethernet in in-vehicle networks where nodes communicate with each other via Ethernet switches, effectively communicating like a point-to-point connection.

in vehicles [48]. Over time, the technology has been fully standardised to prevent proprietary implementations and ensure seamless interoperability across vehicles, networking components, and software systems. These standards have played a crucial role in fostering collaboration within the automotive industry and enabling the widespread adoption of Ethernet as a reliable in-vehicle networking solution.

One of the key advantages of Automotive Ethernet is its ability to support significantly higher baud rates compared to legacy automotive networking technologies like CAN, LIN, or FlexRay that are required for modern vehicles as shown in Table 2.3 [49]. This increased bandwidth enables the seamless integration of advanced features such as high-resolution cameras for ADAS (where cameras send about 500-3,500 Mbps of video frame data), lidar and radar sensors for autonomous driving, and real-time multimedia streaming for infotainment systems (transmitting video data from storage media to the head unit requires 5-25 megabits per second (Mbps).) [50].

The growing number of sensors, cameras, and advanced features in vehicles generates significant data traffic between the ECUs that manage various subsystems. Unlike shared bus topologies, Ethernet uses a symmetric point-to-point architecture, allowing each pair of nodes to communicate at the full designated network bandwidth, such as 100 Mbps or more. This design enables the total aggregate bandwidth across all connections to scale into the gigabits-per-second (Gbps) range, accommodating the high data demands of modern in-vehicle networks [51].

Moreover, Automotive Ethernet facilitates the reuse and adaptation of Internet Protocol (IP)-based software technologies from other industries, such as telecommunications and enterprise IT. This allows for streamlined development and integration of functionalities such as functional safety, over-the-air (OTA) updates, and robust cybersecurity measures. By leveraging established IP frameworks, Automotive Ethernet also supports scalable and interoperable system architectures, paving the way for future developments in connected and autonomous vehicles [52].

However, some current disadvantages of Automotive Ethernet in vehicles include the need for further validation by manufacturers, as it is relatively new compared to older technologies like CAN. Additionally, Ethernet incurs higher costs and adds more weight to the system arising from the complexity of the network controller and the protocol switches/gateways that need to be deployed for interconnection and interoperability.

In summary, Automotive Ethernet combines high-speed communication, cost-efficiency, and adaptability, making it an essential enabler for modern and next-generation vehicle networks. Its ability to bridge automotive needs with established Ethernet technologies provides a strong foundation for innovation in intelligent and connected mobility.

2.2.3 Other networks: FlexRay, LIN, MOST

This subsection briefly overviews other networks used in vehicular environments and their respective purposes.

2.2.3.1 FlexRay

FlexRay is a sophisticated automotive network communications protocol originally developed by the FlexRay Consortium, a group formed to address the increasing complexity of on-board vehicle computing. FlexRay was designed to offer faster and more reliable data transmission compared to earlier protocols such as CAN

and Time-Triggered Protocol (TTP). Although the consortium disbanded in 2009, FlexRay was formalised into a set of international standards: ISO 17458-1 through ISO 17458-5. These ISO standards ensure the continued adoption and consistent application of FlexRay in various industries [53].

FlexRay operates as a communication bus that supports high data rates (up to 10 Mbps), fault tolerance (features two independent data channels for fault tolerance, allowing communication to continue at reduced bandwidth if one channel becomes inoperative), and deterministic data exchange, making it ideal for demanding automotive and aeronautic applications. It explicitly supports both bus and star-based network topologies and uses unshielded twisted pair cabling to connect nodes. Differential signalling on each wire pair minimises the impact of external noise on the network, eliminating the need for costly shielding [54, 55].

It employs a time-triggered communication cycle, which is divided into two key segments:

- **Static Segment:** Allocated for time-triggered communications, ensuring predictable and deterministic data exchange for safety-critical systems such as braking and steering.
- **Dynamic Segment:** Reserved for event-triggered communications, allowing more flexible transmission of non-critical data such as infotainment system updates or sensor readings.

This dual-segment architecture enables FlexRay to balance deterministic timing requirements with the need for flexibility in handling dynamic events, making it well-suited to advanced applications in autonomous vehicles, electronic stability control systems, and adaptive cruise control. It is mainly used in the aeronautic and automotive sectors.

2.2.3.2 Local Interconnect Network

Local Interconnect Network (LIN) acts as a supplement to the CAN bus. LIN 1.0 was introduced by the LIN Consortium in 1999, which included prominent members such as BMW, Volkswagen, Audi, Volvo, Mercedes-Benz, Volcano Automotive, and Motorola. The bus offers lower performance and reliability compared to other in-vehicle networks like CAN but stands out as a highly cost-effective option for applications where speed and fault tolerance are not critical. It is commonly used in

vehicles to control non-critical systems such as windows, windshield wipers, and air conditioning.

A typical LIN cluster consists of one master and up to 15 slave nodes, connected by a single wire (+ground) that supports data rates of 1–20 kbit/s over a maximum bus length of 40 meters. The network uses time-triggered scheduling to guarantee latency, supports variable data lengths (2, 4, or 8 bytes), and includes features like error detection, checksums, and configuration capabilities. LIN operates on a 12V or 24V single-wire system (as opposed to CAN which operates on 5V dual wire), with a physical layer based on the ISO 9141 (K-line) standard. Additionally, it supports sleep mode and wakeup functionality, making it energy-efficient. Most modern vehicles now incorporate over 10 LIN nodes, leveraging its low-cost design for non-critical functions [56].

Today, the LIN bus has become a de facto standard in nearly all modern vehicles, serving a variety of applications across different systems. In steering wheels, it is used for features like cruise control, wipers, climate control, and radios. Comfort systems rely on LIN for sensors that monitor temperature, sunlight, light levels, and humidity. In the powertrain, it supports position, speed, and pressure sensors, while in the engine, it operates small motors and cooling fan motors.

LIN also plays a significant role in air conditioning systems, controlling motors, and the control panel, as these systems are often complex. For doors, it enables the operation of side mirrors, windows, seat controls, and locks. Similarly, it is used in seats for position motors and pressure sensors, and other areas like window wipers, rain sensors, headlights, and airflow systems [9, 57].

The use of LIN bus transceivers is expected to grow steadily at an annual rate of 5–10%, mirroring the projected growth of CAN bus transceivers. This anticipated growth aligns with the rise of Automotive Ethernet, highlighting a prevailing belief that LIN and CAN will continue to play crucial roles in automotive systems for many years to come.

2.2.3.3 Media Oriented Systems Transport

MOST (Media Oriented Systems Transport) is a high-speed multimedia network technology designed for the automotive industry, supporting both in-vehicle and external applications. It utilises a serial bus with a daisy-chain or ring topology and synchronous serial communication to transmit audio, video, voice, and data signals.

These signals are carried via plastic optical fibre (POF) in MOST25 and MOST150 implementations or through electrical conductors in MOST50 and MOST150.

MOST technology uses a single interconnection to transport audio, video, data, and control information, serving as a versatile solution for multimedia networking. It supports various physical layers, including fibre optic, unshielded twisted pair (UTP), and coaxial cables, and offers data transfer speeds of 25, 50, and 150 Mbps. This makes it an ideal connectivity backbone for integrating a wide range of multimedia interfaces within a network [58]. MOST technology has been widely adopted by car manufacturers around the world, including Audi, BMW, General Motors, Honda, Hyundai, Jaguar, Lancia, Land Rover, Mercedes-Benz, Porsche, Toyota, Volkswagen, SAAB, Škoda, SEAT, and Volvo.

The MOST network can support up to 64 devices in a ring configuration. Its plug-and-play functionality allows devices to be easily added or removed without disrupting the network. Additionally, MOST networks can be configured in various topologies, including a virtual star network. For safety-critical applications, redundant double-ring configurations are employed to enhance reliability.

MOST technology can connect multiple consumer devices directly to one another or within a network configuration, utilising either optical or electrical physical layers. As a synchronous network, it delivers excellent Quality of Service (QoS), ensuring seamless connectivity for audio and video streaming. The MOST specification outlines the protocol, as well as the hardware and software layers required, to enable efficient and cost-effective transport of control, real-time, and packet data through a single physical medium [8].

Within the MOST network, one device is designated as the timing master, responsible for continuously supplying the ring with MOST frames. Each frame begins with a preamble, which is used by the other devices, known as timing followers, to maintain synchronization [59].

2.3 Security challenges in automotive networks

Automotive networks have been continuously evolving to enhance the wide range of services provided by modern vehicles, as discussed in earlier sections [60]. However, despite these advancements, security was not a primary focus during the initial development of these networks.

With the increasing connectivity of vehicles to external systems, numerous potential vulnerabilities have emerged. Access points such as Bluetooth modules,

telematics units, and the onboard diagnostic (OBD-II) port expose vehicles to threats, effectively transforming them into "computers on wheels." These connectivity features make vehicles susceptible to cyberattacks, much like traditional computing systems [50, 61].

To address these vulnerabilities, robust security mechanisms, such as firewalls, are essential to safeguard vehicles and ensure they can shift to a safe operating mode when faced with remote attacks. A critical component of such systems is the ability to detect intrusions effectively. This involves monitoring the networks that carry crucial system messages, such as those for the engine control unit, braking systems, or airbag deployment.

In most vehicles, CAN still serves as the backbone for transmitting vital information between ECUs responsible for critical operations. As such, developing efficient IDS for CAN networks is imperative to protect these essential communications and ensure vehicle safety and security.

CAN has established itself as the de facto industry standard for communication between critical ECUs in vehicles, primarily due to its robust protocol specification and lower cost. Its key advantages, as discussed earlier, include the ability to reduce overall wiring within a vehicle, thereby lowering both manufacturing costs and vehicle weight. Additionally, CAN is highly reliable and efficient, making it well-suited for transmitting critical information across the network of ECUs [62].

However, despite its strengths, CAN was not developed with security as a priority, as it originated in the late 1980s and early 1990s when cybersecurity concerns in automotive systems were minimal. One significant limitation is its broadcast-based transmission mechanism, which transmits messages to all ECUs on the network. While this approach simplifies communication, it also creates a significant vulnerability: a compromised ECU can intercept and monitor all CAN traffic. This unrestricted access can result in a loss of user privacy and expose sensitive vehicle data to malicious actors.

Moreover, CAN lacks built-in encryption and authentication mechanisms, leaving it vulnerable to spoofing and other external attacks. Without a way to verify the authenticity of messages, a malicious actor could inject harmful or deceptive data into the network, potentially disrupting critical systems like brakes, airbags, or engine control [63].

The vulnerabilities in the CAN protocol have been exposed in real-world scenarios, most notably in the 2015 hacking of a Jeep Cherokee. In this high-profile case, ethical hackers demonstrated how they could remotely exploit a security flaw to gain control

over the vehicle's critical systems, including steering, braking, and acceleration. This alarming demonstration not only highlighted the lack of cybersecurity in modern vehicles but also prompted the recall of over 1.4 million vehicles by Fiat Chrysler Automobiles (FCA). This incident served as a wake-up call to the automotive industry about the potential consequences of unsecured vehicle networks [64].

Such incidents are not isolated and are likely to become more frequent as vehicle connectivity continues to grow. For instance, researchers have uncovered vulnerabilities in other vehicles, including Teslas and GM's OnStar system. In one instance, security experts showed how they could manipulate Tesla's autopilot system by introducing false sensor readings through a simulated attack. Similarly, vulnerabilities in the OnStar RemoteLink app were exploited to remotely unlock and start a vehicle. These examples underscore the critical need for robust security measures to protect modern vehicles from malicious attacks [65, 66].

As vehicles integrate advanced technologies such as 5G connectivity, over-the-air software updates, and Internet of Things (IoT) devices, the attack surface expands significantly. For example, connected infotainment systems, smartphone integration, and V2X communication offer convenience but also open new pathways for hackers. A compromised infotainment system could lead to unauthorised access to other critical ECUs via the CAN network, potentially endangering passenger safety for example:

- Flood the network (DoS attack) and prevent legitimate communication.
- Send false data (Spoofing)
- Suppress valid data (Silencing ECUs by occupying the bus)
- Take over control of critical functions (Tampered ECU).

Some of the methods an attacker might use to exploit each system and the potential impacts of such attacks are shown below:

- **Safety Systems (ABS, Airbag Deployment, Emergency Brake Assist)**
 - **ABS:**
 - * Spoofing or disabling ABS signals, leading to malfunction.
 - * Sending false wheel speed data.
 - **Airbag Deployment:**

- * Injection of false crash signals, triggering unnecessary deployment.
- * Suppressing valid crash signals, preventing deployment in real accidents.
- **Emergency Brake Assist:**
 - * Blocking or delaying brake assist commands.
 - * Falsely triggering emergency braking without driver input.
- **Body Control Systems (Doors, Lights, Climate Control)**
 - **Doors:**
 - * Unauthorized unlocking or locking.
 - **Lights:**
 - * Manipulating headlights, brake lights, or interior lights.
 - * Flashing lights to distract the driver.
 - **Climate Control:**
 - * Altering temperature settings or disabling defrosters.
- **Telematics Systems (Infotainment, Radio, GPS)**
 - **Driver Distraction:**
 - * Changing radio stations, increasing volume, or modifying infotainment display.
 - **Data Leakage:**
 - * Accessing driver data such as contacts or messages.
 - **Gateway Vulnerability:**
 - * Using telematics as an entry point to attack safety or powertrain systems.
- **Powertrain Systems (Engine Control, Transmission Control, Oil Supply Control)**
 - **Engine Control:**
 - * Injecting false RPM, throttle position, or sensor data.
 - * Causing engine misfire or shutdown.
 - **Transmission Control:**

- * Sending incorrect gear shift commands.
- **Oil Supply Control:**
 - * Sending false oil temperature or pressure data.
- **Throttle Manipulation:**
 - * Spoofed throttle commands leading to unintended acceleration.

Figure 2.7 shows how an affected ECU can disrupt the network traffic in a CAN-based system.

Without proactive measures, future incidents could have more severe consequences, particularly as vehicles progress toward higher levels of automation. For instance, the potential for cyberattacks on autonomous vehicles could result in hackers gaining control of navigation systems, rerouting vehicles, or even disabling critical safety features. To address these risks, OEMs must prioritise the development of secure protocols, robust intrusion detection systems, and continuous vulnerability assessments to ensure the safety and security of increasingly connected and autonomous vehicles.

CAN is currently deployed in more than 100 million vehicles worldwide and remains a cornerstone of in-vehicle communication systems. Despite the emergence of newer technologies like Automotive Ethernet, CAN is likely to remain an integral part of future in-vehicle networks, often working alongside these newer protocols to support a hybrid network architecture. Its widespread adoption and compatibility with existing systems make it a practical and cost-effective solution for a wide range of applications, from powertrain communication to body control systems [67].

However, the vulnerabilities inherent in CAN, such as its lack of built-in security features and susceptibility to attacks due to its broadcast-based transmission mechanism, highlight the urgent need for robust cybersecurity solutions. As vehicles evolve towards higher levels of automation, particularly Level 4 (high automation) and Level 5 (full automation), the safety and reliability of in-vehicle networks become paramount. These advanced automation levels rely heavily on the seamless and secure communication of critical data between sensors, ECUs, and actuators to ensure safe navigation, obstacle detection, and real-time decision-making [68, 69].

To address these challenges and support the transition to higher levels of automation, it is essential to develop and deploy effective IDS tailored to CAN networks. These systems must monitor network traffic for anomalies, detect malicious activities in real time, and ensure the integrity and confidentiality of data within the network. Additionally, integrating advanced security features, such as message authentication,

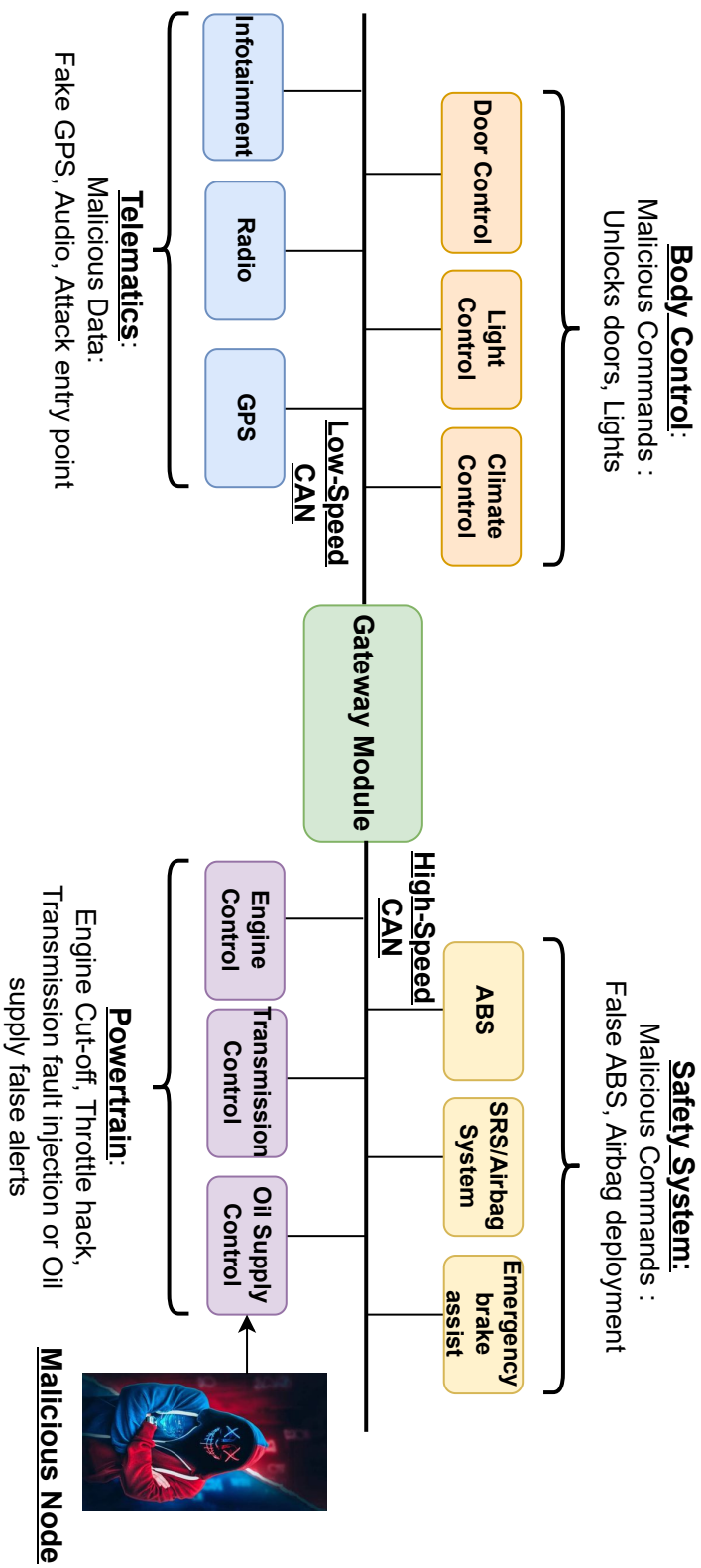


Fig. 2.7 System diagram of the CAN network (featuring components connected to high-speed/low-speed CAN) shows how an affected ECU can cause network tampering.

encryption, and network segmentation, can further enhance the resilience of CAN networks.

These systems, however, must be optimised not only for classification accuracy but also for several other critical factors, such as deployment latency, energy efficiency, and resource utilisation, particularly in resource-constrained vehicular environments. These considerations are essential to ensure the practical feasibility and real-time performance of these systems, as discussed in Section 1.3. Proper optimisation ensures that the systems can operate reliably under the stringent conditions typically encountered in automotive settings, including limited computational resources and power constraints.

By addressing these vulnerabilities, the automotive industry can ensure that CAN remains a secure and reliable backbone for communication in future vehicles, enabling the safe deployment of autonomous driving technologies and safeguarding the trust of users in an increasingly connected and automated world [70].

2.4 Intrusion detection systems for CAN

Given the vulnerabilities of automotive networks, particularly CAN, which facilitates critical in-vehicle communication, it is essential to integrate these networks with robust intrusion detection and prevention mechanisms. Over the past few years, this issue has garnered increasing attention, leading to early research efforts focused primarily on developing IDSs. These systems aim to identify malicious activities within automotive networks, which could then serve as a trigger for intrusion prevention systems (IPS) to mitigate potential threats in real-time. The availability of public attack datasets has significantly contributed to the advancement of intrusion detection research, as it enables the training and evaluation of machine learning and rule-based detection models. Consequently, a substantial portion of the literature has been dedicated to enhancing detection techniques, leading to the development of more sophisticated and efficient solutions. Prevention solutions have also been proposed in the literature; however, their development is limited by the need for proprietary vehicle information, which is often inaccessible and not openly available.

In this subsection, the initial approaches proposed for intrusion detection in automotive networks are reviewed, highlighting their strengths and limitations. More advanced and recent methodologies from the research literature are subsequently discussed. These approaches leverage machine learning, anomaly detection, and hybrid security frameworks to improve detection accuracy and response times.

2.4.1 Early Intrusion Detection Systems (2015-2020)

Researchers have explored numerous techniques for detecting intrusions in CAN-based systems. These methods can be broadly categorised into **flow-based**, **payload-based**, and **hybrid** approaches, which combine elements of both flow and payload analysis [71].

Flow-based approaches focus on network-level characteristics such as message frequency, inter-message intervals, and transmission patterns to detect anomalies that may indicate unauthorised activities [72, 73]. In [74], authors propose a novel algorithm for extracting real-time model parameters of the CAN bus and introduce SAIDuCANT, a specification-based IDS. SAIDuCANT leverages anomaly-based supervised learning, using the real-time model as input to detect intrusions. The system's effectiveness is evaluated using real CAN logs collected from two passenger vehicles, as well as an open-source CAN dataset derived from real-world scenarios. Experimental results demonstrate that SAIDuCANT successfully detects data injection attacks while maintaining low false positive rates. In [28] authors propose the Clock Offset-based Intrusion Detection System (COIDS) to monitor in-vehicle networks and detect potential intrusions. Specifically, they measure and utilise the clock offset of the transmitter ECU to create unique fingerprints for each ECU. These fingerprints are then used to establish a baseline of normal ECU clock behaviour through an active learning technique. They measure deviations in the clock offset and if the deviation surpasses a predefined positive or negative threshold, COIDS flags the event as a potential intrusion. Additionally, a sequential change-point detection technique is applied to pinpoint the exact time of the intrusion. In [75] authors propose a novel intrusion detection system for the CAN bus that detects anomalies by leveraging the bus's intrinsic delay characteristics and monitoring the difference in signal arrival times at both ends of the bus. By analysing deviations from expected timing behaviour, these techniques can flag suspicious traffic patterns without requiring access to message contents.

Payload-based techniques, on the other hand, examine the contents of CAN frames to detect irregularities in instruction sequences or data values [76, 77]. In [76] authors propose an algorithm that detects anomalies in payload sequences across different ID classes by computing the Hamming distance between consecutive payloads. Its detection performance is evaluated through experiments using real CAN traffic collected from an unmodified, licensed vehicle. These methods are particularly

useful for identifying attacks that manipulate control commands or inject malicious messages into the network.

Hybrid approaches integrate both timing-based and payload-based analysis to provide a more comprehensive intrusion detection mechanism [78–80]. In [81] authors propose an intrusion detection mechanism that leverages Bloom filtering to analyse frame periodicity based on message identifiers and specific data-field segments, enabling the detection of potential replay or modification attacks. By leveraging both sources of information, these methods can create unique fingerprints of transmitting and receiving ECUs, enhancing detection accuracy.

Other notable approaches include fingerprint-based methods that utilise low-level electrical signal characteristics, such as voltage variations and timing deviations, about message contents to identify anomalies. Large deviations from established patterns can indicate potential intrusions [82, 83]. Similarly, parameter monitoring-based techniques extract message-level timing and frequency characteristics for specific CAN fields. For example, these approaches can analyse offset ratios or the timing behaviour of remote transmission frames to detect deviations that suggest malicious activity [84].

Another approach involves measuring the entropy of network traffic to identify abnormal conditions. In [85] authors propose a novel information entropy-based method that utilises a fixed number of messages as sliding windows. By refining the sliding window strategy and optimising decision conditions, the approach enhances detection accuracy while reducing the false positive rate. Entropy-based methods detect sudden shifts in the randomness or predictability of CAN messages, which may indicate unauthorised modifications to normal communication patterns.

Other approaches like updating the protocol of the CAN controller [86, 87] have also been presented in the literature. Some flavours of prevention systems were also part of the research works like in [88] authors propose a novel algorithm to detect and recover from message spoofing attacks that target the integrity of the CAN bus. By utilising the predictable runtime behaviour of CAN message frames during the recovery process, the authors take advantage of the CAN bus’s error-handling capability (bus-off state) to implement a reboot-based recovery of the compromised network node.

These foundational approaches have been further extended by machine learning-based techniques, including traditional classification algorithms, deep learning models, and sequential analysis methods. By leveraging data-driven insights, these techniques aim to improve detection accuracy and adaptability to emerging attack vectors.

2.4.2 Machine Learning-based IDSs for CAN

Most ML based intrusion detection approaches fall under the hybrid scheme, leveraging both message frame content and timing/frequency characteristics to enhance detection accuracy. These techniques combine statistical analysis with ML models to identify patterns indicative of various attack types.

Several research efforts have explored different ML architectures for CAN intrusion detection. Alshammari et al. [89] employ a Support Vector Machine (SVM) classifier, utilising offset ratios and time intervals between remote requests to detect Denial-of-Service (DoS) and fuzzy attacks. Similarly, Yang et al. [90] explore tree-based classification methods, including decision trees and random forests, using CAN IDs and data segment contents as input features.

Deep learning approaches have also been investigated. Song et al. [18] propose a reduced InceptionNet architecture for IDS, leveraging deep convolutional neural networks (CNNs). Their model achieves over 99% detection accuracy across DoS, fuzzing, and spoofing attacks using a dataset captured from an actual vehicle. This dataset has been made publicly available to facilitate further research. Given its real-world attack coverage and authentic CAN messages, this dataset is adopted as the baseline for training and evaluating the models that are discussed in this thesis.

Other advanced ML-based IDSs include Generative Adversarial Network (GAN)-based approaches. Seo et al. [91] introduce GIDS, a GAN-trained IDS that utilizes CAN IDs as input features and achieves 97.5% average accuracy in detecting DoS, fuzzy, and spoofing attacks.

Recent advancements have introduced more complex ML architectures to further enhance detection accuracy. These include:

- Temporal convolution with global attention mechanisms [92]
- Unsupervised learning models combining CNNs and long short-term memory (LSTM) networks [93].
- Gated Recurrent Units (GRU)-based networks [94]
- CNN-based temporal dependency approaches

These methods incorporate various input features, including CAN ID, ID + Data Field, or entire CAN frames, as summarized in Table 4.1.

Beyond IDS, some approaches extend into IPSs. De Souza et al. [95] propose an iForest anomaly detection algorithm as an IPS, targeting fuzzing and spoofing

Table 2.4 Input features used by the ML-based IDSs & IPSs proposed in the research literature.

Models	Input Features Used	Year
GIDS [91]	CAN ID	2018
DCNN [18]	CAN ID	2020
Rec-CNN [32]	CAN ID	2022
iForest [95]	Data Field	2021
MLIDS [31]	CAN ID + Data Field	2020
NovelADS [93]	CAN ID + Data Field	2022
TCAN-IDS [92]	CAN ID + Data Field	2022
MTH-IDS [23]	CAN ID + Data Field	2021
GRU [94]	CAN ID + Data Field + DLC	2022

attacks (RPM & Gear). The system marks anomalous messages as errors, preventing their propagation to other ECUs. However, this approach has a drawback—false positives or DoS attacks can result in multiple legitimate messages being dropped, potentially impacting vehicle functionality.

In practical deployments, most ML-based IDS models rely on high-performance GPUs to achieve real-time detection rates when processing CAN traffic. These models are benchmarked using CAN datasets, evaluating detection accuracy and processing latency in a batch-style operation.

This thesis examines techniques to design and integrate ML accelerators as a standard ECU peripheral and subsequently as a network extension within a standard ECU. To achieve this, techniques are developed to optimise performance to achieve high detection accuracy and throughput with significantly lower power consumption compared to traditional GPU implementations. The proposed systems' efficacy is evaluated using the following metrics:

- Detection accuracy evaluation
- Processing latency analysis (both batch and per-message)
- Power consumption assessment for real-world feasibility

2.4.2.1 Public attack datasets

Most of the aforementioned ML-based IDSs for CAN have been evaluated using the two widely adopted attack datasets discussed below. The first dataset used widely is the **open Car Hacking dataset** [18], which contains normal data along

with separate data files for DoS, Fuzzing, and Spoofing (RPM/Gear) attacks. Each attack file in the dataset was generated by logging data from the OBD-II port of real vehicles while targeted injections were performed at run-time. Each attack was sustained for 3 to 5 seconds, resulting in a dataset comprising 30 to 40 minutes of total CAN traffic. The attacks exhibit distinct signatures, as shown below:

- **DoS Attack:** Repeatedly injecting messages with the dominant CAN ID ‘0000’ every 0.3 milliseconds to overwhelm the network.
- **Fuzzing Attack:** Injecting messages with completely random CAN ID and DATA values every 0.5 milliseconds to disrupt normal network operation.
- **Spoofing Attack (RPM/Gear):** Injecting messages with specific CAN IDs associated with RPM or gear information every 1 millisecond to imitate or alter legitimate data.

The second dataset used to evaluate the ML-based IDS models and demonstrate their generalisation capability is the **Survival Analysis Dataset for Automotive IDS** [96]. This dataset primarily focuses on three types of attacks that can severely disrupt in-vehicle functions or escalate attack impact and damage: Flooding, Fuzzing, and Malfunction attacks. Like the Car Hacking dataset, this dataset includes one file for normal traffic and three separate files for each attack. The attack packets are injected for 5 seconds at 20-second intervals for each type of attack. To enhance generalisability, these three attacks were carried out on three different vehicles: the Hyundai Sonata, Kia Soul, and Chevrolet Spark. The specific attack signatures within this dataset are as follows:

- **Flooding Attack:** This attack resembles the DoS attack in the Car Hacking dataset, with both datasets injecting dominant CAN ID 0x0000. However, this dataset introduces complexity by including multiple message IDs with payloads set to all zeros across all 8 bytes, making the attack pattern slightly more intricate.
- **Fuzzing attack:** Similar to the Car Hacking dataset, this attack involves injecting messages with random CAN IDs and payloads on the bus every 0.3 milliseconds.
- **Malfunction attack:** The malfunction attack targets specific CAN IDs similar to the spoofing attack in the Car Hacking dataset. In this attack, manipulation

of the data field is combined with the injection of messages using randomly selected CAN IDs to disrupt the vehicle's functionality. IDs specific to each vehicle were chosen to launch the attack.

Both these datasets have been widely used in the research literature for developing and evaluating various IDS approaches. Consequently, these datasets are leveraged to rigorously test, validate, and demonstrate the effectiveness of the IDS model.

2.5 Deployment Challenges for IDSs

Most of the IDS solutions discussed in the previous subsections primarily evaluate their performance based on detection accuracy metrics using publicly available datasets. While accuracy is a crucial factor, real-world deployment presents additional challenges that are often overlooked.

Only a few approaches consider the practical feasibility of deploying IDS solutions in actual automotive environments. Many fail to assess their systems in terms of:

- **Processing latency:** Ensuring real-time detection without introducing delays that could impact vehicle functionality.
- **Energy consumption:** Evaluating the power efficiency of IDS implementations, which is critical for embedded automotive ECUs.
- **Resource utilisation:** Analyzing computational requirements, including CPU, memory, and hardware accelerator dependencies.
- **Scalability and integration:** Designing IDS solutions that can be seamlessly integrated into existing vehicle architectures without requiring significant modifications.

To bridge this gap, a detailed discussion of these deployment constraints is presented, incorporating quantitative metrics to highlight the real-world feasibility and limitations of current IDS approaches.

2.5.1 Understanding Real-Time Constraints in CAN Bus

The inter-frame arrival time (or message inter-arrival time) in CAN is determined by the bus speed and network load. A real-time IDS must process each message before the next one arrives to prevent backlog and message drops.

The time between successive CAN frames, T_{arrival} , is given by:

$$T_{\text{arrival}} = \frac{N_{\text{bits}}}{R_{\text{bus}}} \quad (2.1)$$

where:

- T_{arrival} = time between successive CAN frames (seconds)
- N_{bits} = number of bits in a CAN frame (typically 128-160 bits including overhead)
- R_{bus} = CAN bus bitrate (bps, typically 125 Kbps – 1 Mbps)

For a 1 Mbps CAN bus:

$$T_{\text{arrival}} = \frac{128}{10^6} = 128 \text{ } \mu\text{s} \text{ (minimum)} \quad (2.2)$$

For a 500 Kbps CAN bus:

$$T_{\text{arrival}} = \frac{128}{5 \times 10^5} = 256 \text{ } \mu\text{s} \quad (2.3)$$

Thus, the IDS processing latency per message, T_{IDS} , must satisfy:

$$T_{\text{IDS}} \leq T_{\text{arrival}} \quad (2.4)$$

The total IDS processing time in most cases consists of:

- **Preprocessing Time** (T_{pre}): Extracting features like timing, payload, and identifiers.
- **Classification Time** (T_{class}): Running ML or rule-based detection.
- **Decision & Response Time** (T_{decision}): Flagging an anomaly and initiating a response.

Thus, the total processing latency per message:

$$T_{\text{IDS}} = T_{\text{pre}} + T_{\text{class}} + T_{\text{decision}} \quad (2.5)$$

For real-time operation:

$$T_{\text{pre}} + T_{\text{class}} + T_{\text{decision}} \leq T_{\text{arrival}} \quad (2.6)$$

CAN Bus Speed	Min. IFAT (T_{arrival})	IDS Processing Latency (T_{IDS})
1 Mbps	128 μs	$T_{\text{IDS}} \leq 128 \mu\text{s}$
500 Kbps	256 μs	$T_{\text{IDS}} \leq 256 \mu\text{s}$
250 Kbps	512 μs	$T_{\text{IDS}} \leq 512 \mu\text{s}$
125 Kbps	1024 μs	$T_{\text{IDS}} \leq 1024 \mu\text{s}$

Table 2.5 The table shows different CAN bus speeds and the corresponding maximum permissible IDS processing latency.

Table 2.5 shows the minimum IDS processing times for different CAN bus speeds. Some key takeaways here are:

- If T_{IDS} exceeds T_{arrival} , the IDS introduces a backlog, leading to delayed message processing and potential packet drops.
- Efficient IDS designs optimise T_{class} using lightweight ML models, hardware acceleration (e.g., FPGA, TPU), or parallel processing.

For an IDS to be real-time capable, it must process each message within the inter-frame arrival time dictated by the CAN bus speed. Optimised architectures and hardware acceleration are key to ensuring IDS meets stringent latency requirements.

2.5.2 Energy & Resource Constraints for Real-Time IDS in Automotive Networks

In addition to the crucial accuracy and latency requirement, in-vehicle IDS deployments are constrained by the limited energy budget for electronic systems and the computational constraints of typical ECU platforms.

2.5.2.1 Energy Constraints for IDS Deployment

Modern ECUs operate under strict power budgets due to the limited energy availability in vehicles, especially in electric vehicles where energy efficiency is critical.

Table 2.6 shows different ECUs based on vehicle types and the energy budget associated with each ECU [97]. An IDS must consume less than the available power budget of the target ECU while processing the messages in real-time. Some important takeaways from the above table are:

ECU Type	Power Budget (W)	Processor Type
Low-end ECU	< 10W	Microcontroller (MCU)
Mid-range ECU	10W - 100W	ARM Cortex-A/R/M
High-end ECU (ADAS, IVI)	> 100W	SoC (e.g., NVIDIA Orin)

Table 2.6 The table shows different ECU designs and their respective energy budgets.

ECU Type	CPU Performance (DMIPS)	RAM (MB)
Low-end ECU	50 - 200 DMIPS	64 - 256MB
Mid-range ECU	500 - 2000 DMIPS	256MB - 2GB
High-end ECU (ADAS, IVI)	5000+ DMIPS	4GB+

Table 2.7 The table shows different ECU types and the corresponding resource budgets (compute performance, memory) available with them.

- Lightweight ML models (SVM, Decision Trees, etc.) are better suited for low-power ECUs.
- Deep Learning-based IDS (CNN, LSTM, GANs) require hardware acceleration/optimisations (e.g., FPGA, TPU) to reduce power consumption.

2.5.2.2 Resource Constraints for IDS Deployment

Another challenge for IDS integration, irrespective of the deployment strategy, is the resource constraints of standard ECU platforms. When an IDS is integrated within an ECU, it shares critical compute resources with the primary ECU function(s); hence the IDS model should have limited computational complexity and memory footprint to minimise the impact on the critical functions. Table 2.7 provides an overview of the resource availability across different types of ECUs [98, 99]. In the ideal case where IDS is implemented as a standalone ECU, it must still adhere to the overall energy constraints and the resource constraints imposed by the specific platform. An IDS deployment faces limited resource and energy budget due to the system constraints; however, it must be capable of processing the messages in real-time satisfying the functional requirement of IDS.

To enable the efficient deployment of IDS in vehicles, various optimisations are required for ML models to reduce computational overhead while maintaining high accuracy. Such optimisations rely on reducing the numeric precision and optimising the operations performed to make them more suited for resource-constrained micro-

controller platforms. These optimisations also allow the ML models to be adapted for energy-efficient hardware accelerators such as FPGAs, which can significantly enhance performance and reduce power consumption. The following subsection discusses the case for using (hybrid) FPGAs as an ECU platform for embedding scalable IDS capabilities.

2.6 Field Programmable Gate Arrays

FPGAs are specialised hardware accelerators known for their flexibility, reconfigurability, and exceptional energy efficiency, particularly when compared to other accelerator technologies such as GPUs. Unlike GPUs, which are designed for parallel processing of general-purpose tasks, FPGAs offer a customisable architecture, allowing users to tailor hardware to specific tasks for optimal performance. This unique ability to reconfigure hardware dynamically makes FPGAs highly attractive for a variety of applications, particularly those where energy efficiency and low latency are paramount.

Over the years, FPGAs have found widespread use in areas such as space exploration, telecommunications, and network infrastructure, where their energy efficiency and ability to handle specialised tasks have made them indispensable [100–102]. In recent years, they have also emerged as an ideal solution for energy-efficient ML inference. Unlike traditional computing systems, FPGAs can execute ML algorithms with significantly lower power consumption, making them ideal for edge devices and battery-operated systems [103].

The following sections provide an overview of FPGA-based approaches for automotive systems that have been proposed to address scalable real-time latency-sensitive processing requirements and functional consolidation schemes. The specific benefits of FPGA-based approaches for ultra-low-latency ML inference accelerators presented in the research literature are also discussed. With the increasing demand for both performance and power efficiency in modern automotive systems, FPGAs are increasingly becoming a platform of choice for next-generation vehicular electronic systems.

2.6.1 Machine Learning on FPGAs

While GPUs have traditionally been the dominant choice for machine learning inference, FPGAs have emerged as a compelling alternative, offering significant ad-

vantages in energy efficiency, customisation, and real-time processing. However, early implementations of deep learning accelerators on FPGAs faced notable challenges, primarily due to limited on-chip memory resources. This constraint necessitated frequent off-chip memory accesses to store model weights and biases, leading to performance bottlenecks and increased latency compared to GPU-based implementations [104].

Additionally, early FPGA-based deep learning accelerators required highly customised and optimised designs, demanding significant hardware expertise to manage efficient data movement between off-chip memory and compute units. The need for extensive low-level programming and hardware design knowledge created a steep learning curve, limiting widespread adoption.

Recent advancements in FPGA-based deep learning inference have significantly reduced these barriers, making FPGAs more accessible while still delivering considerable performance and energy efficiency gains over GPUs [105–107]. Modern FPGA toolchains and frameworks have introduced high-level synthesis (HLS) techniques, enabling seamless mapping of deep learning models onto FPGA hardware with minimal manual intervention.

Notable frameworks such as DNN-Weaver [108], LUTNET [109], and FINN [110] facilitate efficient deployment by converting high-level representations of neural networks into hardware-optimised implementations. These toolchains leverage techniques such as model pruning, quantisation, and hardware-aware optimisations to reduce computational complexity and memory footprint, thereby improving both performance and power efficiency.

A key advantage of FPGA-based deep learning acceleration is the ability to customise data representation formats (e.g., floating-point, fixed-point, or binary precision) to strike an optimal balance between accuracy, latency, and resource utilisation. For instance, frameworks like FINN allow aggressive quantisation, reducing precision down to 1-bit weights and activations while applying post-quantisation fine-tuning to mitigate accuracy loss. These optimisations generate highly efficient, low-power implementations suitable for real-time embedded applications.

With a moderate level of hardware expertise, designers can optimise every layer of a neural network—including weights, biases, and activation functions—to create domain-specific, high-performance FPGA accelerators. These accelerators can be integrated as slave co-processors within an embedded system or deployed as standalone inference engines, offering scalable and energy-efficient solutions for AI-driven applications across automotive, robotics, and edge computing domains.

Alternatively, vendor-provided toolchains such as Vitis-AI from Xilinx [111] streamline the deployment of machine learning models on FPGAs by mapping pre-trained networks onto a selection of pre-composed, configurable Deep Learning Processing Units [112]. This approach significantly lowers the barrier to entry for using FPGA-based ML accelerators, eliminating the need for extensive hardware expertise while still offering notable performance and energy efficiency benefits.

The Vitis-AI flow facilitates model quantisation and (optionally) fine-tunes pre-trained networks to generate hardware-optimised accelerator implementations. It also provides the necessary software stack, including drivers and APIs, to seamlessly offload computations from the host code to the FPGA-based accelerator. While this flow does not offer the same level of fine-grained optimization as fully custom implementations generated through frameworks like FINN, it remains a compelling alternative to GPU-based inference. Specifically, at `int8` precision, Vitis-AI achieves energy efficiency comparable to custom FPGA implementations while maintaining flexibility and ease of use.

For the initial techniques discussed in this thesis, the Vitis-AI flow is leveraged to design and deploy the proposed IDS accelerators on hybrid Zynq Ultrascale+ SoCs, integrating FPGA-based ML acceleration within an embedded platform. This approach enables rapid prototyping while ensuring high performance and energy efficiency.

For subsequent designs, to further optimise latency, resource utilisation, and energy efficiency, the FINN toolflow is adopted to generate highly customised FPGA accelerators [110]. FINN's ability to produce ultra-low latency, fully quantized (e.g., 1-bit precision) models allows us to achieve significant reductions in inference time while maintaining high classification accuracy. By leveraging FINN's hardware-aware optimisations, dedicated IDS accelerators that are tailored for real-time, high-throughput vehicular intrusion detection can be generated.

The specific designs discussed in this thesis aim to establish an integrated IDS-ECU architecture that consolidates intrusion detection functions within dedicated slave accelerators, operating alongside the primary ECU function on the same chip/package. This design closely mimics commercial off-the-shelf ECU architectures while offering additional security and performance benefits. The FPGA-based accelerators execute our QCNN, QMLP, and QLSTM models directly on incoming CAN messages, ensuring full isolation from the host processor. The experimental results demonstrate that this integrated architecture provides unique advantages in

terms of latency, energy efficiency, and detection accuracy across multiple attack vectors, validated using publicly available automotive cybersecurity attack datasets.

2.6.2 FPGAs for automotive applications

FPGA-based deployments are increasingly gaining traction in the automotive industry, driven by their ability to accelerate real-time, computationally intensive algorithms while maintaining energy efficiency. Prior research has demonstrated their effectiveness in a range of applications, including vision-based ADAS, fault detection, and high-performance Ethernet-based gateways [113–116]. By leveraging their parallel processing capabilities and reconfigurable logic, FPGAs enable low-latency execution of critical tasks, making them well-suited for safety-critical automotive applications.

Beyond performance improvements, FPGAs have also been explored for enhancing ECU functionality and security. Researchers have investigated their use in network interface extensions to bolster the security of in-vehicle communication networks [117]. This includes integrating cryptographic acceleration, secure boot mechanisms, and anomaly detection frameworks to mitigate cyber threats targeting modern connected vehicles [118].

To ensure greater adaptability while maintaining compliance with industry standards, FPGA-based ECU architectures have been examined as a means to support custom functionalities while adhering to AUTOSAR specifications [119]. Automotive suppliers and vendors have further explored hybrid FPGA-based ECUs that combine traditional microcontroller-based architectures with FPGA acceleration to enhance functional consolidation, improve system reliability, and enable seamless scalability for next-generation automotive platforms [120]. Recently, Porsche has demonstrated the viability of FPGAs as part of automotive ECU platforms by integrating them into a real-time, heterogeneous multiprocessor system for managing motor control, battery operations, and charging functions. The FPGA complements multicore processors by handling complex real-time computations and enabling scalable, flexible ECU architectures beyond traditional microcontroller-based solutions [121].

One particularly promising area of FPGA adoption is network security, where their real-time processing capabilities have been leveraged to enhance anomaly detection techniques. FPGA platforms have demonstrated improvements in the latency and accuracy of intrusion detection mechanisms, including bit-timing analysis [122], transmission timing analysis [123], and probabilistic attack models for detecting CAN

bus spoofing attacks [124]. These approaches benefit from FPGA-based parallelism to analyse high-throughput vehicular network traffic without imposing excessive computational overhead.

As the computational demands of modern vehicles continue to rise—with increasing autonomy, sensor fusion, and high-speed data processing requirements—hybrid FPGA-based ECUs are poised to play an increasingly critical role in the automotive industry. By offering a balance between performance, flexibility, and real-time processing capabilities, FPGAs are emerging as a key enabler for the next generation of intelligent, secure, and high-performance vehicular systems.

2.7 Summary

In this background chapter, the problem statement introduced earlier is expanded, providing a detailed discussion of the motivation behind this thesis. This chapter aims to equip the reader with the necessary context and foundational knowledge to fully understand the subsequent chapters. By presenting relevant background information, it ensures a smooth transition into the core contributions of this work.

Subsequent chapters present the technical contributions of this thesis, with multiple quantised neural network-based IDS models that progressively enhance computational performance, latency and energy consumption for the CAN bus along with specific integration optimisations at the ECU. The approaches are evaluated in terms of classification accuracy when evaluated on public attack datasets and for implementation constraints such as detection latency, energy consumption and on-device resource utilisation.

Chapter 3

Quantised feed-forward NN as IDSs for Controller Area Network

3.1 Introduction & Contributions

This chapter investigates model compression techniques for machine learning-based IDS models and their impact on inference accuracy, latency, and energy consumption of the IDS models. Quantisation is applied to weights, activations, and biases of convolutional neural network-based and multilayer perceptron-based intrusion detection models. By exploring multiple models, the efficacy of quantised architectures in detecting anomalies within CAN traffic while maintaining efficiency and real-time constraints can be evaluated. A quantised Deep-CNN model is used as the starting point, building on the most successful intrusion detection approach at the time, using CAN IDs as the primary input features [18]. Iterative optimisation of this model led to a more compact variant (through transfer learning and layer-wise pruning), significantly reducing the resource cost and per-message processing latency without sacrificing detection accuracy for individual attack vectors.

A key limitation of the CNN-based IDS model was its inability to directly incorporate the payload of CAN messages as input features. To achieve high classification accuracy using payload data, the CNN approach required analysing larger temporal windows of messages, making it impractical for real-time intrusion detection due to increased computational complexity and latency as discussed in the section 2.2. To address this limitation, quantised MLP models were explored as a solution, based on their ability to model time series sequences as shown in the literature. Unlike CNNs, these models could learn intrinsic features when both CAN IDs and payload data

were used as input features (with trivial pre-processing steps), allowing for more comprehensive anomaly detection. Combining quantisation with dataflow optimisations, the MLP-based IDS achieved state-of-the-art classification accuracy across multiple attack vectors while maintaining low resource and energy overheads for real-time deployment. In both cases, their classification performance, computational efficiency, and compliance were evaluated with the deployment constraints outlined in the previous section. Additionally, the integration architecture of each of the proposed approaches within a model ECU is presented, highlighting the trade-offs between CNN and MLP models in terms of latency, resource utilisation, and detection capabilities.

Subsequently, the limit of reducing precision is investigated with the MLP model to arrive at a highly quantised version. This highly quantised version achieves exceptionally high accuracy in classifying multiple attack vectors and benign messages. This custom quantised MLP model also significantly reduces IDS processing latency, bringing it below the inter-frame arrival time for a CAN bus operating at 1 Mbps. This enables real-time, line-rate intrusion detection in CAN networks.

The rest of the chapter is organised as follows: The chapter begins with an introduction to quantised Deep-CNN models (QdCNN & MA-QCNN) in Sections 3.2 & 3.3, followed by an in-depth evaluation and discussion of their integration into a model ECU. Further, Section 3.4 discusses the design and development of the quantised MLP-based IDS (QMLP) along with its performance evaluation and validation as a coupled accelerator to the model ECU. Section 3.5 summarises the above three model implementations and discusses the improvements in accuracy, latency, and energy consumption. Finally, through design iterations, a highly compressed custom quantised MLP accelerator (CQMLP) architecture is achieved which is presented with its evaluation as the coupled accelerator in Section 3.6. In each case, the performance of the proposed IDS is compared against state-of-the-art methods in the literature to highlight the trade-offs. This chapter is concluded in Section 3.7.

3.2 Baseline Model: Quantised deep-CNN

3.2.1 CNN-based IDS

To determine the starting ML model for testing, different network architectures with varying complexity were analysed to find a baseline architecture. High inference accuracy and model complexity were used as parameters to evaluate different models. Research works in the literature guided the architectures explored in this phase, with a strong focus on deep-CNN models as they were found to be particularly successful at detecting attacks on CAN [18]. The brute-force constrained model exploration led to a CNN-based architecture as the baseline model. Sequences of n successive CAN IDs were used as the input feature from which the model extracted optimal features to learn. Additionally, the model exploration was also guided by the network layers supported by AMD's Vitis-AI framework, which was used to compile the model to the hardware accelerator [111]. This model is referred to as **QdCNN** throughout the thesis.

The model consists of 8 **Conv2D** layers implemented with 40, 80, 120, 160, 200, 240, 256 & 512 filters at each layer, each filter having a dimension of 3×3 . The time-series data composed of n sequential CAN IDs is fed as input to the first convolutional layer with 40 filters. Subsequent layer(s) operate on the output of the previous layer, with increasing complexity to extract detailed features. Batch Normalisation and Dropout layers were used between the convolutional layers to prevent over-fitting and to improve the learning efficiency during the training phase. The output layer uses a **softmax** function at the output of the final dense layer to estimate the probability of an infected message. The model is defined in TensorFlow using standard TF functions and nodes.

3.2.2 Dataset and Training

The open Car Hacking dataset is used for training the QdCNN model and to test its performance. The dataset provides a labelled set of normal and attack messages which were captured via the Onboard Diagnostic (OBD) port in an actual vehicle, with attack messages injected in real-time. The dataset includes DoS, Fuzzing, and Spoofing message injections allowing us to validate the detection accuracy across these different attacks. An extract from the dataset is shown in table 3.1, showing the CAN ID (ID field), control field, and the actual data segment. The dataset is split as 80:15:5 for training, validation, and testing respectively, allocating the large section to training and optimisation of the quantised network. The performance of

Table 3.1 An extract from the open Car hacking dataset which is used for the testing and evaluation.

Time	ID	DLC	Data
...			
1478198376.389427	0316	8	05,21,68,09,21,21,00,6f
1478198376.389636	018f	8	fe,5b,00,00,00,3c,00,00
1478198376.389864	0260	8	19,21,22,30,08,8e,6d,3a
...			

the model on the validation set during training ensures that it is not over-fitting on any of the attacks.

The dataset is processed before training to mimic the CAN message data the model will obtain as its input when integrated into the ECU. To achieve this, each message is converted to the equivalent binary value (modelling reception from the CAN interface) from which the IDs are extracted. To generate the FIFO-style operation, groups of n sequential IDs are stacked to form the input tensor for the IDS. The value of n is set as 4 for the DoS and Fuzzing attacks, 8 for Spoofing attacks for optimal performance based on the observation from the design space exploration step. Each layer of this stack forms the combined input shape for the first layer, which is reshaped to feed into the exact channels for training and testing.

To train the model, the Adam optimiser with binary cross-entropy loss function was employed. The learning rate was set to 0.0001 to allow for slower learning which aids in reducing loss of accuracy when quantising the pre-trained model [125]. The training was performed on a workstation class machine with an Intel i9-9820X and an Nvidia 2080Ti GPU. The trained model was exported as a ‘.h5’ checkpoint for quantisation/optimisation by the Vitis-AI flow.

3.2.3 IDS Integration Architecture: QdCNN & MA-QCNN

Given their similarity, the IDS integration of both CNN models introduced in this chapter is discussed in this section.

3.2.3.1 Mapping models to the DPU

The trained checkpoint file in full floating-point representation is fed as the input to the Vitis-AI framework along with the training and validation sets from the dataset. The framework executes multiple optimisation passes, quantising the weights,

activation, and biases at each pass and then evaluating the resulting accuracy of the resultant model to determine the steps for the next pass. This flow converts all parameters and operations to 8-bit (INT8) format, with minimal loss in inference accuracy. The quantised model is further fine-tuned through a quantisation-aware optimisation step that improved the inference accuracy of the quantised model, particularly on the fuzzing dataset for the QdCNN model. Once optimized, the Vitis flow compiles the model to a DPU-executable which incorporates the (quantised) weights, activations, and instruction sequences for executing the model on the DPU. Simultaneously, it also generates a hardware project with the DPU integrated as a slave peripheral of the Zynq PS along with the run-time libraries and API wrappers for the petalinux operating system. The DPU uses three master and one slave AXI interface to move instructions and data (inputs/weights/biases/activations) between the PS, off-chip DRAM, and the DPU internal memory to help maximise the peak performance. Further, the network interface is integrated into the PS the Vivado flow is used to generate the boot image for the Zynq device. For the experiments, the image is transferred to an SD card the device is set up to boot from the SD card image. The QdCNN model is implemented on all available DPU configurations to evaluate the scalability and performance of the model across all possible configurations as this was the baseline model. For the MA-QCNN model, through experiments the best DPU configuration is identified and the results are reported based on that deployment.

3.2.3.2 Integration with Zynq Ultrascale+

Figure 3.1 shows the proposed ECU architecture of the IDS-enabled ECU on a Xilinx Zynq Ultrascale+ device for the QdCNN & MA-QCNN models. The Zynq device integrates capable ARM processors connected to a host of hardened memory-mapped peripheral logic and interface protocols within the processing system (PS) section of the device. Any custom peripheral can be integrated into the programmable logic (PL) region and wired up the PS using high or low-performance Advanced eXtensible Interface (AXI) ports and can be accessed as memory-mapped devices from the software application on the processor. As shown in figure 3.1, an off-the-shelf CAN controller IP is used in the PL which implements the CAN and CAN-FD variants [126]. Though the hardened CAN controller block in the PS could be used in this experiment, the PL integration provides a pathway for updating to future network standards. The PL also instantiates a Xilinx DPU block as the dedicated

ML accelerator which can be configured and controlled using the Vitis AI Runtime (VART) APIs from the PS [112]. The standard Vitis-AI flow is used to automatically generate the DPU IP along with wrappers, interconnects, and the runtime elements from the high-level ML model described in TensorFlow (see Sec. 3.2.1). Both the models fetch CAN message information from the PS over the AXI interface when ready and provide the data back to the software using an interrupt-based readback, allowing the invoking task to run in a non-blocking fashion. Figure 3.1 also shows the packet flow in the receive direction within the ECU. When the CAN interface receives a valid CAN packet, the software task on the PS reads the message like in every ECU application to process the information and take appropriate actions. Additionally, the DPU task sends the packet information to the IDS accelerator to detect any malicious nature that could be embedded in the message sequence. To achieve this, the software task managing the IDS allocates a n 16-bit (or 32-bit in case of extended frames) FIFO-style buffer which is used as the input memory location for the DPU. At the reception of a new packet, the software task extracts the ID bits, updates the buffer, and offloads the computation to the DPU using the non-blocking `execute` command; the DPU runs the model on the packet information and interrupts the PS with a completion status. This scheme allows for a seamless integration of the models with the normal event/time-triggered tasks to be executed by the ECU functions.

3.2.4 Experimental evaluation

The QdCNN model is the initial step towards highly quantised ML models as IDS engines for automotive CAN. To evaluate the impact of quantisation, the pre-quantised model is implemented on an Nvidia RTX A6000 GPU and an Intel i9-10900 CPU mimicking traditional IDS deployment methods at native floating point representations. A PyTorch implementation (pre-quantisation) of the model is also tested on the A53 ARM cores of the PS, modelling a typical dedicated software-driven IDS implementation on an automotive-grade processor.

For the proposed hardware-offloaded QdCNN, the model was deployed on an AXI-connected DPU IP on the Zynq ZU7EV device using AMD’s Vitis-AI flow. A B4096 DPU is chosen with a single compute core and single execution thread to determine the baseline performance and is used to compare against competing approaches on alternate platforms and in the literature. The FPGA accelerator (DPU) uses a 300 MHz interface clock and a 600 MHz DSP core clock, while the

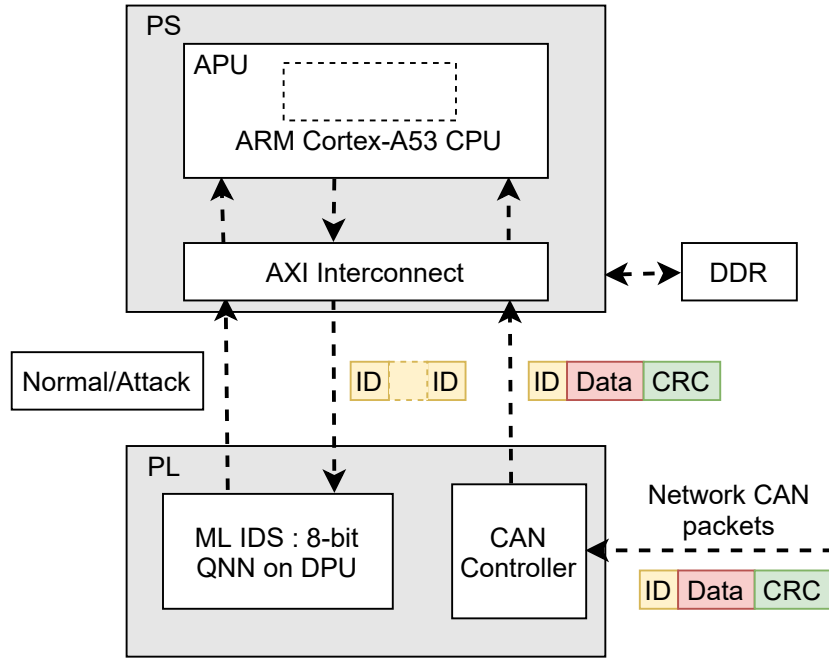


Fig. 3.1 The proposed IDS-ECU architecture with the DPU accelerator attached as a standard peripheral to the processor

ARM PS is clocked at 1.2 GHz. The Core i9 CPU has a base clock of 2.8 GHz. The accuracy of inference is quantified by evaluating precision, recall, F1 score, false positive rate (FPR), and false negative rate (FNR) for both pre-quantised and quantised variants (on CPU/GPU and FPGA respectively). Additionally, the processing latency and power consumption for a single execution cycle (inference cost for a message) are captured on all platforms to quantify the overheads for each CAN message. Subsequently, the scalability of the DPU-based IDS is studied by quantifying the inference latency, resource overheads, and power consumption across different DPU configurations. All measurements are averaged over multiple runs, each using 50000 messages from the test split of the dataset.

3.2.5 Accuracy

Table 3.2 captures the precision, recall, F1 score, FPR, and FNR of the QdCNN model compared to the floating point golden model (pre-Q in table) on the GPU. The results show that the quantised model has nearly identical inference accuracy across different attack vectors, with notable improvement on the Fuzzy attack.

Table 3.3 compares the accuracy metrics of the proposed QdCNN model on the DPU against state-of-the-art works discussed in literature: GIDS [91], DCNN [18]

Table 3.2 Inference accuracy metrics of the model pre- and post-quantization on the four datasets

Attack	Model	Precision	Recall	F1	FPR(%)	FNR(%)
DoS	Pre-Q	99.97	100	99.98	0.01	0
	QdCNN	99.97	100	99.98	0.01	0
Fuzzy	Pre-Q	99.82	76.35	86.52	0.04	23.65
	QdCNN	99.38	98.51	98.94	0.18	1.49
RPM-Spoof	Pre-Q	97.14	98.72	97.93	1.05	1.28
	QdCNN	97.64	98.41	98.03	0.86	1.59
Gear-Spoof	Pre-Q	97.55	99.22	98.38	1.58	0.78
	QdCNN	97.62	99.40	98.50	1.54	0.6

& iForest [95]. Among these, the DCNN model is the highest-performing model in terms of inference accuracy metrics. The proposed QdCNN model offers nearly identical detection performance as the DCNN model in the DoS ($\pm 0.03\%$) and Fuzzing attacks ($< 0.6\%$) while falling slightly short ($\approx 2.2\%$) in spoofing attacks. Across the other models, it can be observed that the quantised QdCNN model is on par or better across all metrics for DoS and Fuzzing attacks, with 1.4% performance dip in the spoofing attack.

Table 3.4 shows the confusion matrix resulting from the evaluation of the QdCNN model on the Zynq ECU. The confusion matrix captures the performance of the quantised model beyond the raw accuracy numbers of the model. As inferred from the comparison, the model performs quite well for DoS and Fuzzing attacks but suffers from slightly higher classification errors in the case of spoofing attacks. This performance however could be further improved through focused fine-tuning within the Vitis-AI post-quantisation flow.

3.2.6 Inference Latency, Power & Resource consumption

The per-message inference latency of the QdCNN model is quantified on the proposed ECU architecture and compared against native precision models on other platforms: ARM CPU, desktop-class Intel i9-10900 CPU, and a Nvidia RTX A6000 GPU. These observations are also compared against competing GPU/Raspberry Pi implementations reported in the literature. In each case, the time taken by the inference model is quantified by measuring the start and end times just before and after the execution calls are made. The latency is measured using back-to-back invocations of the ML model (as soon as the current execution is completed) and averaged over 50000

Table 3.3 Accuracy metric comparison of the QdCNN FPGA accelerator to other state-of-the-art methods

Attack	Model	Precision	Recall	F1	FNR
DoS	GIDS	96.8	99.6	98.1	-
	DCNN	100	99.89	99.95	0.13%
	iForest	-	-	-	-
	QdCNN (DPU)	99.97	100	99.98	0%
Fuzzy	GIDS	97.3	99.5	98.3	-
	DCNN	99.95	99.65	99.80	0.5%
	iForest	95.07	99.93	97.44	-
	QdCNN (DPU)	99.38	98.51	98.94	1.49%
RPM-Spoof	GIDS	98.3	99	98.6	-
	DCNN	99.99	99.94	99.96	0.11%
	iForest	98.97	100	99.48	-
	QdCNN (DPU)	97.64	98.41	98.03	1.59%
Gear-Spoof	GIDS	98.1	96.5	97.2	-
	DCNN	99.99	99.89	99.94	0.12%
	iForest	94.79	100	97.33	-
	QdCNN (DPU)	97.60	99.40	98.50	0.6%

Table 3.4 Confusion matrix capturing the classification results of the QdCNN on the DoS, Fuzzy, and Spoofing attacks

Attack	Message Type	Predicted Attack	Predicted Normal
DoS	True Attack	16269	0
	True Normal	5	33726
Fuzzy	True Attack	11012	166
	True Normal	69	38753
RPM-Spoof	True Attack	13080	211
	True Normal	316	36393
Gear-Spoof	True Attack	19299	117
	True Normal	471	30113

runs. The results can be observed in table 3.5. The ARM CPU incurs 22 ms for inference on average making it a poor candidate for line-rate IDS in CAN networks. While the GPU and Intel i9 CPU offer improved latency, the tightly coupled DPU accelerator in the proposed ECU architecture offers over $2\times$ improvement compared to the GPU in the case of per-message execution. It should be noted that the PCIe transactions involved in GPU are very inefficient for short data transfers, which

Table 3.5 Per-message latency and measured power of QdCNN accelerator on different platforms.

Platform	Latency (ms)	Measured Power	
		Idle (W)	Active (W)
Zynq-PS	22.32	2.58	2.96
i9-10900	1.8	35	152
RTX A6000	1.1	22	96
DPU Accelerator (Zynq-PL)	0.53	4.37	5.76

impacts the latency in the case of the GPU model. Switching to batch mode on the GPU (executing all 50000 messages at once), it is possible to reduce the average execution to 0.195 ms; however, it requires a significant number of CAN messages to be accumulated (batch size) before any potential threat can be identified, limiting its application as a line rate IDS.

The active power consumption of each platform is also quantified while executing the inference model. Here, the idle power draw is monitored on each platform with standard background tasks active and the active power consumption is measured when the model is invoked to arrive at the power consumed by the platform while executing the model. In the case of the GPU, the Nvidia plugin (nvidia-smi) was used to measure the GPU power consumption directly allowing it to be isolated from the host machine's power consumption. On the Zynq device, the on-device power rail monitors were used to measure accurate device-level power consumption through the PMBus APIs. Table 3.5 tabulates the power consumption measured across the different target architectures. It was observed that the idle power consumption on the Zynq ZCU104 development board with the PS running Petalinux was 2.58 W, while loading the PL region with the DPU bitstream raised the idle power consumption to 4.37 W. The ARM core on the PS consumes 2.96 W while executing the model, while offloading the execution to the DPU results in total power consumption of 5.76 W by the Zynq device (and peripherals). The proposed DPU offload in the ECU architecture can thus reduce the power consumption by 94% while achieving lower per-message latency and nearly identical detection performance as a dedicated GPU-based IDS. Note that the offload causes negligible overhead in the ARM cores (as opposed to software IDS deployment, consuming an estimated 4.5 W during single core operation on a Raspberry-Pi 4 at full utilisation), allowing IDS capability to be integrated with any critical ECU function on the Zynq device with minimal effort.

Table 3.6 Per-message latency comparison against state-of-the-art IDSs (GPU/Raspberry Pi) reported in literature

	Latency (ms)
MLIDS [31]	275 (per CAN message, GPU)
GIDS [91]	5.89 (block of 64 CAN frames, GPU)
DCNN [18]	5 (block of 29 CAN frames, GPU)
MTH-IDS [23]	0.574 (per CAN message, Raspberry Pi)
QdCNN on DPU	0.53 (per CAN message, DPU on Zynq FPGA)

Table 3.7 Resource utilisation breakdown for the PL modules of the proposed CAN IDS-ECU (XCZU7EV)

Node	LUT	FF	BRAM/URAM	DSP
DPU (B4096)	48313	97508	84.5/ 46	690
CAN FD	1932	2110	6/0	0
Overall	64660	117562	116.5/46	704
(% usage)	(28.06%)	(25.51%)	(37.34%/47.92%)	(40.74%)

The latency of the proposed IDS-ECU architecture is compared against other implementation platforms discussed in literature [18, 23, 31, 91] in Table 3.6. The DCNN & GIDS approaches operate on a block of 29 & 64 CAN messages at each execution, consuming 5 ms & 5.89 ms to process the block on an Nvidia Tesla K80 GPU & GTX 1080 GPUs respectively. On average, this leads to a processing latency of 0.17 ms & 0.09 ms per message, at the expense of higher waiting time for the arrival of the batch of messages. In [31], the authors measure the per-message latency of 275 ms for their model, when executed on an Nvidia Titan X (GTX 200) GPU. In [23], the authors report a per-message latency of 0.574 ms when their lightweight multi-model stacked IDS is executed on a Raspberry Pi 3 device. In comparison, the integrated IDS-ECU incurs 0.53 ms per-message latency, and in combination with the high detection accuracy across multiple attack modes, lower power consumption (vs GPU) and the ability to tightly couple IDS with ECU function, lends itself as the ideal architecture for real-time IDS on CAN networks.

The resource consumption of the custom modules on the PL part of the proposed IDS-ECU on the Zynq XCZU7EV device is shown in table 3.7. It can be seen that the largest DPU model (B4096) that can be deployed on this device consumes most of the resources as expected; however, it still leaves enough resources for other custom accelerators which could be integrated into the ECU to offload parts of its tasks.

Table 3.8 Performance scaling of the QdCNN model using different DPU configurations: impact on per message latency (ms), resource (%), & power consumption (W)

DPU	Latency (ms)	% Resource Utilisation					Power (W)	
		LUT	FF	DSP	BRAM	URAM	Idle	Active
B512	0.81	12.46	7.96	6.37	13.94	12.5	2.87	3.84
B800	0.70	13.55	9.47	9.09	15.54	29.17	3.14	4.15
B1024	0.81	15.55	11.03	12.62	26.76	14.58	3.17	4.29
B1152	0.63	14.67	10.83	12.27	18.91	33.33	3.16	4.42
B1600	0.70	17.14	13.57	18.06	29.97	33.33	3.40	4.83
B2304	0.63	18.82	15.71	24.42	36.54	37.50	3.68	4.98
B3136	0.58	20.84	18.16	31.71	42.95	41.67	3.91	5.46
B4096	0.53	23.25	22.22	39.93	47.12	47.92	4.37	5.76

3.2.7 Performance scaling using alternate DPU configurations

Vitis-AI offers a range of DPU configurations allowing the designer to trade-off performance for resource and power consumption. The different DPU configurations tune the amount of processing and input-output parallelism, incurring higher resources and power consumption for higher performance. Table 3.8 shows various performance/utilisation/power metrics when the QdCNN model is run on different DPU configurations on the Zynq XCZU7EV device. The increased performance of B4096 DPU improves the latency per message by 34.65% while utilising more DSP and distributed memory (BRAM/URAM) blocks and increasing the power consumption from 3.84 W to 5.76 W compared to the baseline B512 DPU. From table 3.8, B1152 DPU offers the best trade-off, achieving nearly 22% reduction in message latency at under 6% increase in DSP utilisation and around 15% increase in power consumption compared to the B512 DPU.

The scalability of the IDS-ECU approach is also evaluated by quantifying the effects of pipelined execution on the DPU (threaded mode). As discussed, the baseline model used a single compute core model within the DPU, which when executed in the threaded mode, utilises the internal pipelined operation to evaluate multiple CAN messages in parallel and hence does not incur additional resources. However, with multiple compute cores enabled within the DPU (incurring more resources), higher performance benefits are observed when combined with threaded operation. Figure 3.2 shows the latency per message for different DPU configurations, each with two compute cores operating in threaded mode. As the number of threads increases from 1 to 3, a considerable reduction is observed in the processing latency

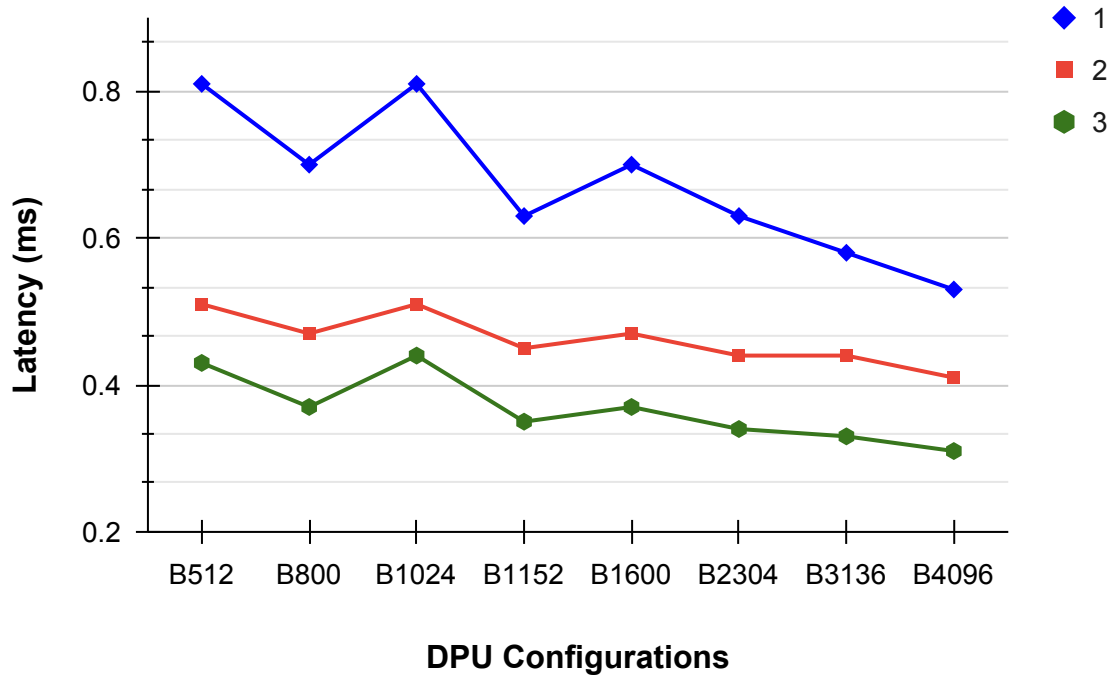


Fig. 3.2 Per-message latency of the QdCNN model on all DPU configurations after enabling multiple threads from 1 to 3.

from the graph, with the B4096 DPU achieving per message processing latency of 0.31 ms (41.5% reduction) over the single compute core single-threaded mode configuration at 22% increase in LUT consumption (over single core mode) while consuming 8.07 W during execution. Increasing threads beyond 3 does not bring any significant performance advantage from our tests and is hence not shown. As before, the B1152 achieves the best trade-off, achieving a per-message latency of 0.35 ms at a slightly higher power consumption of 5.58 W (3 threads), leaving behind enough resources for an additional DPU (isolated IDS on Gateway) or for other dedicated accelerators.

The QdCNN model served as the baseline implementation, demonstrating that QNNs can achieve high classification accuracy for a range of attack types, while simultaneously meeting deployment constraints through an FPGA-based pipelined implementation. In the next section, the QdCNN model is optimised using layer pruning techniques, and transfer learning is leveraged to construct a model capable of detecting multiple attack vectors concurrently.

3.3 QdCNN optimisation: Transfer learning & Pruning

3.3.1 MA-QCNN model

To improve the message processing latency and attack detection efficiency of the QdCNN model, a further optimised CNN-based IDS architecture was developed. This new model leverages transfer learning to enable the detection of multiple attack types simultaneously. To further reduce the static energy consumption of the target platform, the low-power Ultra96-v2 platform was chosen as the target, which integrates the smaller XCZU3EG FPGA device.

The architecture comprises five `Conv2D` layers with 40, 80, 120, 160, and 200 filters respectively, each utilizing a 3×3 kernel. These are followed by a `Flatten` layer and two `Dense` layers with 32 and 1 units, respectively. The final dense layer employs a `sigmoid` activation function to predict the likelihood of an incoming message being an attack. This model differs from QdCNN by incorporating fewer convolutional layers and introducing a classification-dense layer at the output. It is referred to as MA-QCNN in this chapter.

3.3.2 Dataset & Training

The MA-QCNN model is trained to detect both DoS and Fuzzing attacks from the CAR Hacking dataset using a unified architecture. It takes blocks of four CAN IDs as input features to capture temporal and contextual information from the network traffic. While the training procedure follows a similar pipeline to that of QdCNN, it extends the previous workflow by incorporating inductive transfer learning to enable joint detection of multiple attack types [127].

Initially, the model is trained on the DoS attack data to achieve optimal detection performance. This pre-trained model is then fine-tuned on the Fuzzing dataset, allowing it to generalise effectively across both attack vectors. Finally, the resulting model is evaluated on both DoS and Fuzzing attack scenarios to ensure that the extended training does not lead to performance degradation in either class.

Table 3.9 Inference accuracy metrics of the proposed MA-QCNN model, pre- and post-quantisation on the DoS & Fuzzing attacks.

Attack	Model	Precision	Recall	F1	FPR	FNR
DoS	Pre-Q	99.92	100	99.96	0.04%	0%
	MA-QCNN	99.92	100	99.96	0.04%	0%
Fuzzy	Pre-Q	99.66	98.75	99.2	0.1%	1.25%
	MA-QCNN	99.66	98.78	99.22	0.1%	1.22%

3.3.3 Experimental Evaluation

The model integration and the IDS architecture followed the same steps as previously discussed, with Ultra96-v2 as the target platform. The specific implementation is referred to as MA-QCNN, which should be capable of detecting both DoS & Fuzzing attacks using a single model. This consolidation leads to improved per-message processing latency and power consumption (from the static power consumption), as discussed below.

3.3.4 Accuracy

The pre- and post-quantisation accuracy of the model is verified using the test split of the dataset and is shown in Table 3.9. The post-quantisation fine-tuned model performance showed no visible degradation in inference performance measuring using Precision, Recall, F1 score, FNR & FPR metrics.

The packaged ‘xmodel’ file is executed using the DPU accelerator in the PL with the VART libraries installed on the OS running on the PS. For the experiments, a petalinux-based image was used for the Ultra-96 V2 board on which the PYNQ and VART libraries were installed to use the standard VART/PYNQ APIs. Subsequently, the bitstream corresponding to the DPU accelerator is loaded to the PL from the OS (as an overlay). The B1152 DPU was utilised for most of the performance measurements (unless specified) with the DPU using a 150 MHz interface clock and 300 MHz DSP core clock.

The model performance is compared in terms of inference accuracy metrics and processing latency with the state-of-the-art IDSs and IPSs described in the literature. Performance metrics such as Precision, Recall, F1 score, and FNR are used to compare the inference accuracy of other models where reported and use per-message latency to capture the true latency of the detection approach starting from the arrival of a new CAN message at the interface. The power consumption of the ECU

Table 3.10 Confusion matrix capturing the classification results of the MA-QCNN (on DPU).

Attack	Message Type	Predicted Normal	Predicted Attack
DoS	True Normal	33282	13
	True Attack	0	16705
Fuzzy	True Normal	38784	38
	True Attack	136	11042

while executing the IDS is quantified and compared against the execution of the full-precision model on a GPU. Note that none of the ML-based IDS approaches in the literature report the actual power consumption of their implementations; however, since most models are significantly more complex than the MA-QCNN model, it is assumed that the GPU implementation would be a good baseline for comparison.

3.3.5 Inference Accuracy

Table 3.11 compares the proposed MA-QCNN with the state-of-the-art approaches discussed in the research literature: GIDS [91], DCNN [18], iForest [95], MTH-IDS [23], TCAN-IDS [92] & NovelADS [93]. In the case of MTH-IDS, the authors only report an average accuracy, precision & recall of 99% across all attacks which is identical to our model on the three metrics. In the case of DoS attack, the quantised hardware accelerator performs almost identically to the DCNN, TCAN-IDS, NovelADS, and MLIDS across all metrics and has superior performance than GIDS across all metrics. In the case of fuzzing attack, it is observed that DCNN, MLIDS & NovelADS perform marginally better, with 0.6%, 0.7% & 0.8% better F1 scores respectively. The proposed model is almost identical to TCAN-IDS in case of fuzzing attacks and offers a marginal improvement over iForest and GIDS by 1.7% and 0.9% respectively (F1 scores). Note that in competing techniques, the models are trained afresh for each attack, whereas the proposed model uses a transfer learning approach during training and hence, uses the same weights, biases and activation values for both attacks. Table 3.10 shows the confusion matrix of MA-QCNN on the two attacks. The receiver operating characteristic (ROC) curve is also plotted for the two attacks when using the model on the Ultra-96 V2, which is shown in Figure 3.3. The area under curve (AUC) values for the model were also determined using the test datasets. It is observed that the model achieves an AUC value >0.99 in both attacks with a slightly lower performance in the fuzzing attack.

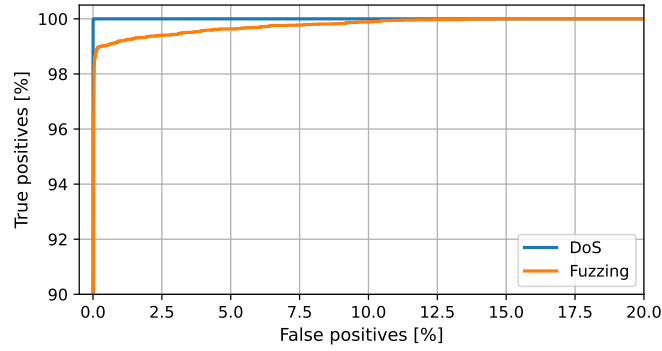


Fig. 3.3 ROC curve of the MA-QCNN model on the DoS and Fuzzing attacks.

Table 3.11 Accuracy metric comparison (%) of the quantised FPGA accelerator (MA-QCNN) against the reported literature.

Attack	Model	Precision	Recall	F1	FNR
DoS	GIDS [91]	96.8	99.6	98.1	-
	DCNN [18]	100	99.89	99.95	0.13
	MLIDS [31]	99.9	100	99.9	-
	NovelADS [93]	99.97	99.91	99.94	-
	TCAN-IDS [92]	100	99.97	99.98	-
	iForest [95]	-	-	-	-
	MA-QCNN (DPU)	99.92	100	99.96	0
Fuzzing	GIDS [91]	97.3	99.5	98.3	-
	DCNN [18]	99.95	99.65	99.80	0.5
	MLIDS [31]	99.9	99.9	99.9	-
	NovelADS [93]	99.99	100	100	-
	TCAN-IDS [92]	99.96	99.89	99.22	-
	iForest [95]	95.07	99.93	97.44	-
	MA-QCNN (DPU)	99.66	98.78	99.22	1.22

3.3.6 Latency, Power consumption & Resource utilisation

The cold-start setup time of IDS is observed to quantify the time from power on to the model being ready. This is an important consideration since many attacks could be initiated during the startup phase of the network and ECUs. In the case of the test platform, it is observed this cold-start latency is 8 seconds when averaged across multiple runs. It should be noted that the test platform uses a slower boot device (SD image) and full Linux OS, whereas typical ECUs rely on non-volatile flash memory and real-time operating systems which could reduce the cold-start latency by up to $6\times$.

Next, the processing latency of the inference datapath is evaluated from the arrival of a new message and compared against competing approaches in the literature. As mentioned before, some approaches perform block-based detection and hence require the block of messages to be available before it is fed as input to the inference engine. This reduces average inference latency per message leveraging the parallelism on the GPUs. However, accumulating 64 messages (for example) consumes nearly 8.3 ms on a 1 Mbps CAN network with 8-byte messages, which should be factored into their worst-case detection latency when averaging. The results are shown in table 3.12, also capturing the execution platform and the block size in case of block-based inference. It is observed that the lightweight model of the proposed approach takes 0.43 ms from the arrival of a message from the CAN interface, when measuring the processing time to the arrival of the last message of the block, achieving a 1.3x speedup over state-of-the-art CAN IDSs. Finally, the model does incur additional latency due to message looping through the ECU and could be reduced further using a CAN interface in the PL feeding the DPU through a FIFO. Also, on a more capable hybrid FPGA, a larger DPU could be configured to have multiple execution units and/or multiple execution threads (as explored with the QdCNN model) to reduce latency further for higher-speed CAN networks at the expense of slightly higher power consumption and device cost.

The power consumption of the IDS-ECU is quantified while performing inference, using the PYNQ-PMBus package to monitor the power rails directly. All power measurements are averaged over 100 inference runs, each evaluating the test set lasting nearly 10 seconds. The Ultra-96 ECU consumes 2 W when the CAN messages are been analysed using the DPU block on the PL. In comparison, the lightweight floating-point model running on the A6000 consumes 54 W while performing inference at similar CAN data rates, which is $27\times$ higher than the entire IDS-ECU on the Ultra-96 device. Other IDS implementations in the literature do not report power measurements; however, the lightweight model on the GPU should be a good baseline, and more complex models discussed in the literature should consume higher power during inference on GPUs. Table 3.13 shows the resource utilisation on the PL, with the design consuming 45% DSP blocks and 57% BRAMs on the XCZU3EG device, leaving sufficient resources for other custom logic. Thus, the IDS-ECU on a hybrid FPGA like an Ultra-96 device offers an ideal mix of detection accuracy, latency, and power consumption, while offering software-driven control of IDS and seamless GPU-like deployment with minimal change to the ECU application, making it a suitable choice for in-vehicle deployment.

Table 3.12 Per-message latency comparison against state-of-the-art IDSs (GPU/Raspberry Pi) reported in the literature.

Models	Latency	Frames	Platform
MLIDS	275 ms	per CAN frame	GTX Titan X
NovelADS	128.7 ms	100 CAN frames	Jetson Nano
GIDS	5.89 ms	64 CAN frames	GTX 1080
DCNN	5 ms	29 CAN frames	Tesla K80
TCAN-IDS	3.4 ms	64 CAN frames	Jetson AGX
MTH-IDS [23]	0.574 ms	per CAN frame	Raspberry Pi 3
MA-QCNN (DPU)	0.43 ms	per CAN frame	Zynq FPGA

Table 3.13 Resource utilisation breakdown for PL accelerator of the proposed CAN IDS (XCZU3EG).

Node	LUT	FF	BRAM	DSP
DPU	30726	48400	123	164
Overall	32648	51347	123	164
(% usage)	(46.2%)	(36.3%)	(56.9%)	(45.5%)

The MA-QCNN model represented a significant advancement by integrating layer pruning and inductive transfer learning strategies, resulting in improved inference speed compared to QdCNN while enabling the detection of multiple attack types within a single unified model. Despite these improvements, both QdCNN and MA-QCNN were constrained to using blocks of CAN IDs as input features and were unable to leverage message payloads during training. However, incorporating payload information alongside CAN IDs is critical for developing more comprehensive and effective IDSs. The following section addresses this limitation using MLP-based IDS architectures that are capable of utilising both CAN IDs and payloads for enhanced detection in automotive CAN networks.

3.4 Quantised MLPs as IDSs

While quantised-CNNs were effective for classification (with CAN IDs as input features) they could not be trained with both IDs and message payloads without significant performance degradation. This led to an exploratory search of different model architectures that can classify attacks using full CAN frames as the input feature. MLPs were found to be more accurate in attack detection (accuracy, false

positives, and false negatives) when using the entire message contents. MLPs also benefit from significantly lower computational complexity and inference latency. It was observed that MLPs could also be trained to detect multiple attack vectors using a single model like the MA-QCNN model. A concatenation of $n = 4$ successive messages (CAN IDs + payload) was chosen as the input feature based on this exploration. The model exploration was also guided by the network layers supported by the Vitis-AI framework.

3.4.1 MLP Model

The model consists of 5 **Dense** layers implemented with the 256, 128, 64, 32 & 1 units at each layer. The time-series data from the input feature buffer is fed as input to the input layer with 40 units. Subsequent layer(s) operate on the output of the previous layer, with increasing complexity to perform classification. Batch Normalisation and Dropout layers were used between the dense layers to prevent over-fitting and to improve the learning efficiency during the training phase. The output layer uses a **sigmoid** function at the output of the final dense layer to estimate the probability of an infected message. The model is defined in TensorFlow using standard TF functions and nodes. The variants of this model are referred to as **QMLP-1** & **QMLP-2** in this chapter. Implementing two different models detecting four different attacks allows them to be detected all on the same platform.

3.4.2 Dataset & Training

As before, the open Car Hacking dataset was used for training the models and to test its performance [128], with the same 80:15:5 training, validation, and testing split. The performance of the model on the validation set during training ensures that it is not over-fitting on any of the attacks. The same pre-processing steps were applied to remap the dataset contents to mimic the CAN message the model will obtain as its input, when integrated into the ECU. The message fields (ID and payload) of each message are packed into INT8 type vectors (instead of binary features used for CNN models), and a sequence of $n = 4$ of these messages form the input feature buffer content for the IDS (block shape of $\{1, 10*n\}$ INT8 data). Each layer of this stack forms the combined input shape for the first layer, which is reshaped to feed into the exact channels for training and testing.

Adam optimiser was used for training with a binary cross-entropy loss function. The learning rate was set to 0.0001 to allow for slower learning which aids in reducing

loss of accuracy when quantising the pre-trained model [125]. For the first MLP, the model is first trained on the DoS attack and subsequently on the fuzzing dataset to improve generalisation across the two attack modes through inductive transfer. This model file is then tested in both the DoS & Fuzzing attacks to ensure there was no performance degradation. The model was trained for 50 epochs with a batch size of 64. The model saves intermediate results at each epoch allowing us to progressively track and integrate early stopping in case of a significant drop in the accuracy. The same training & validation flow is followed for the second MLP that is trained to classify RPM & Gear spoofing attacks. The training was performed on a workstation class machine with an Intel i9-9820X and an Nvidia A6000 GPU. The trained model files were exported as a '.h5' checkpoint for optimisation by the Vitis-AI flow.

3.4.3 IDS Integration Architecture: QMLPs

Figure 3.4 shows the proposed ECU architecture of the IDS-enabled ECU on a Xilinx Zynq Ultrascale+ device. This integration architecture follows that of the QdCNN and MA-QCNN discussed in section 3.2.3. For this model, the CAN interface receives a valid CAN packet, and the software task on the PS reads the message into its buffer (RAM) for the ECU application to process and take appropriate actions according to the tasks' specifications. The isolated IDS task adds the relevant information from the CAN packet to the input feature buffer and uses the integrated VART APIs to activate the IDS. The input feature buffer is organised as a FIFO (depth of 4 messages) to capture the temporal features of adjacent messages on the CAN bus. Two DPU cores are integrated which execute in a concurrent fashion (non-blocking execution mode), allowing the input features to be evaluated for multiple threat vectors simultaneously. The DPU cores then run the model on the packet information and interrupt the PS with a completion status. This scheme allows for a seamless integration with the normal event/time-triggered tasks to be executed by the ECU functions.

3.4.4 Experimental Evaluation

QMLP-1 & QMLP-2 represent the two MLP models used for detecting all the attacks of the open CAR hacking dataset simultaneously. The floating point variants of both models were trained using an Nvidia A6000 GPU prior to quantisation through Vitis-AI. Post quantisation, the lightweight models comprised 53,300 parameters each and were then packed into the 'xmodel' executable format. For deployment, the Zynq

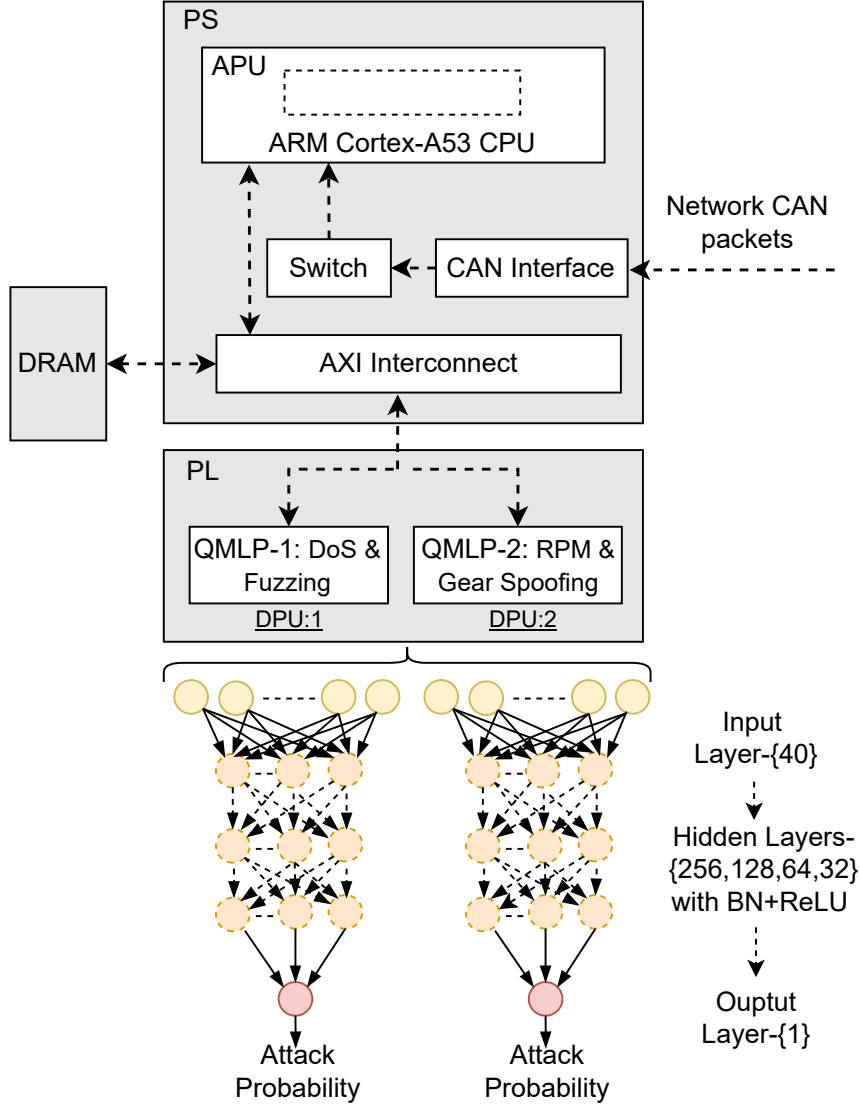


Fig. 3.4 Proposed system architecture of the IDS-ECU with the quantised MLP accelerators on the PL part of the FPGA.

Ultrascale+ ZCU104 development board was chosen as the platform which features an XCZU7EV Ultrascale+ device with quad-core ARM A53 cores and dual-core ARM R5 cores on the PS. The VART APIs are used to deploy the models to the DPUs (in PL) at startup and to invoke test cases at run time. The A53 cores on the PS are configured to run at 1.2 GHz peak. The DPUs use a 600 MHz DSP core clock for their execution and are configured to run concurrently for all the tests at startup.

The inference accuracy was quantified by evaluating precision, recall, and F1 rates as well as the FNR and FPRs for the pre-quantised and quantised variants (on GPU and FPGA respectively). The processing latency and power consumption

Table 3.14 Inference accuracy metrics of QMLP-1, pre and post quantisation {Before and After Fine-tuning (BF & AF respectively)} on the DoS and Fuzzing attacks.

	Precision			Recall			F1-Score			FPR			FNR		
	Pre-Q	Post-Q		Pre-Q	Post-Q		Pre-Q	Post-Q		Pre-Q	Post-Q		Pre-Q	Post-Q	
		BF	AF		BF	AF		BF	AF		BF	AF		BF	AF
DoS	99.81	99.58	99.92	100	100	100	99.91	99.79	99.96	0.09	0.21	0.04	0	0	0
Fuzzing	99.85	91.08	99.86	98.15	99.25	99.67	98.99	94.99	99.76	0.04	2.8	0.04	1.85	0.75	0.33

Table 3.15 Inference accuracy metrics of QMLP-2 pre- and post-quantisation on the RPM & Gear attacks.

Attack	Model	Precision	Recall	F1	FPR	FNR
RPM	Pre-Q	100	100	100	0	0
	QMLP-2	100	100	100	0	0
Gear	Pre-Q	100	100	100	0	0
	QMLP-2	99.90	100	99.95	0.02	0

for performing IDS on each incoming CAN message on the ECU (using the DPU accelerators) are measured and compared against the inference on the A6000 GPU (mimicking a loosely-coupled accelerator) to compare the inference time/power required for each new CAN message on the two platforms. The inference accuracy metrics and message processing latency were compared against state-of-the-art IDSs/IPSSs proposed in the literature. The active power consumption of the models is compared against SOTA ML-IDS approaches in literature where power consumption has been reported.

3.4.5 Accuracy

To test the functional correctness of the models, the pre- & post-quantisation inference performance of both models are compared. A drop is observed in the performance of QMLP-1 on the fuzzing attack which is corrected by the Vitis-AI post-quantisation training flow as shown in table 3.14. Table 3.15 shows the performance of QMLP-2 on the spoofing attacks for which no such degradation is observed.

The inference performance of the QMLPs integrated within the ECU is compared against the state-of-the-art IDSs and IPSs proposed in the literature: GIDS [91], DCNN [18], MLIDS [31], NovelADS [93], TCAN-IDS [92], IForest [95], MTH-IDS [23], GRU [94] and Rec-CNN [32]. F1 score, precision, and recall metrics are used for comparison since most models report the inference performance using these metrics.

Table 3.16 Accuracy metric comparison (%) of the quantised FPGA accelerator (QMLP-1) against the reported literature on the DoS and Fuzzing attacks.

Attack	Model	Precision	Recall	F1	FNR
DoS	GIDS [91]	96.8	99.6	98.1	-
	DCNN [18]	100	99.89	99.95	0.13
	MLIDS [31]	99.9	100	99.9	-
	NovelADS [93]	99.97	99.91	99.94	-
	TCAN-IDS [92]	100	99.97	99.98	-
	iForest [95]	-	-	-	-
	GRU [94]	99.93	99.91	99.92	-
	QMLP-1 (DPU)	99.92	100	99.96	0
Fuzzy	GIDS [91]	97.3	99.5	98.3	-
	DCNN [18]	99.95	99.65	99.80	0.5
	MLIDS [31]	99.9	99.9	99.9	-
	NovelADS [93]	99.99	100	100	-
	TCAN-IDS [92]	99.96	99.89	99.22	-
	iForest [95]	95.07	99.93	97.44	-
	GRU [94]	99.32	99.13	99.22	-
	QMLP-1 (DPU)	99.86	99.67	99.76	0.33

The detailed results are tabulated in tables 3.16 and 3.17 for DoS-Fuzzing attacks and Spoofing attacks respectively. Both MTH-IDS [23] and Rec-CNN [32] report an average accuracy, precision & recall of 99.9% across all attacks which is identical to the proposed model on each of the three metrics. In GRU [94], authors report average precision, recall, and F1 score values of 99.95%, 99.31% & 99.63% respectively for the spoofing attacks. The proposed model achieves the same (MTH-IDS & Rec-CNN) or better (GRU) results on all three metrics for these attacks.

From table 3.16, it can be observed that in the case of DoS attack, the proposed approach offers identical performance to DCNN, MLIDS, NovelADS, TCAN-IDS, and GRU models, while the proposed approach outperforms GIDS model by 1.8% (F1 score). For **fuzzing attack**, the proposed approach offers identical inference accuracy compared to DCNN, MLIDS, NovelADS, TCAN-IDS and performs better than iForest, GIDS, and GRU models by 2.3%, 1.4% and 0.5%, respectively (F1 score). Similarly, from table 3.17, it can be observed that the proposed approach achieves $\geq 99.95\%$ performance across both spoofing attacks across all parameters, and also performs identical to or better than others reported in the literature. It should also be noted that many reported results in the literature rely on retraining the model for each attack and measuring the inference accuracy independently and

Table 3.17 Accuracy metric comparison (%) of the quantised FPGA accelerator (QMLP-2) against the reported literature on the RPM and Gear attacks.

Attack	Model	Precision	Recall	F1	FNR
RPM	GIDS [91]	98.3	99	98.6	-
	DCNN [18]	99.9	99.9	99.9	0.05
	MLIDS [31]	100	100	100	-
	NovelADS [93]	99.9	99.9	99.9	-
	TCAN-IDS [92]	99.9	99.9	99.9	-
	iForest [95]	98.9	100	99.4	-
	QMLP-2 (DPU)	100	100	100	0
Gear	GIDS [91]	98.1	96.5	97.2	-
	DCNN [18]	99.9	99.8	99.9	0.11
	MLIDS [31]	100	100	100	-
	NovelADS [93]	99.8	99.9	99.9	-
	TCAN-IDS [92]	99.9	99.9	99.9	-
	iForest [95]	94.7	100	97.3	-
	QMLP-2 (DPU)	99.9	100	99.95	0

hence require multiple models to be deployed concurrently to detect all the attack vectors [18, 32, 92–94]. In contrast, the proposed approach uses a single integrated IDS-ECU and transfer learning during the training phase to allow every message to be simultaneously evaluated for these different attack signatures.

Tables 3.18 & 3.19 captures the confusion matrix of the QMLP models on the four attacks across the test vectors, showing the raw classification numbers achieved in each case. The misclassifications in the test set can be attributed to scenarios with nearly identical attack and normal message patterns among a stream of messages and/or within the observed block of messages at a given time. Figure 3.5 shows the receiver operating characteristic (ROC) curves of the two QMLP models across the test vectors indicating that the classifiers perform well for all discrimination thresholds. From the ROC curve, it can also be observed that the area under the curve (AUC) values achieved is ≥ 0.99 for all four attacks, showing that the model has a high probability of correctly distinguishing between normal and attack messages.

3.4.6 Inference latency

The per-message processing latency of the models is quantified to show the potential of a tightly coupled IDS with ECU functionality to perform IDS on the message as soon as it arrives. The latency measure includes the time to move data from the

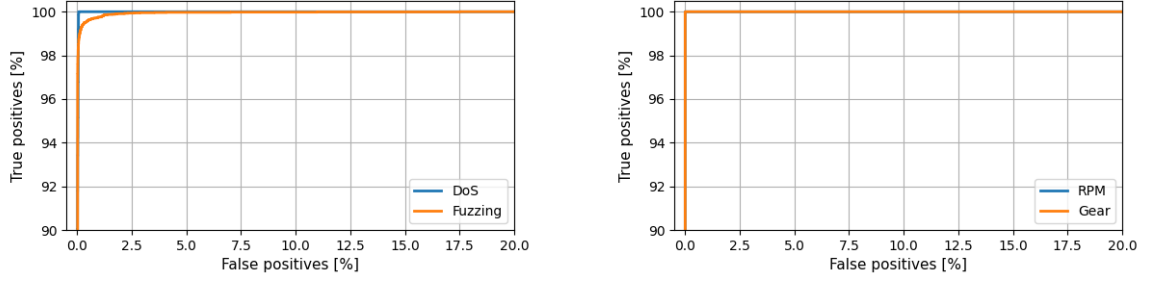


Fig. 3.5 ROC curve of the MA-QMLPs on the four attacks.

Table 3.18 Confusion matrix capturing the classification results of the QMLP-1 model on the DoS and Fuzzing attacks.

Attack	Message Type	Predicted Normal	Predicted Attack
DoS	True Normal	33282	13
	True Attack	0	16705
Fuzzy	True Normal	38806	16
	True Attack	37	11141

Table 3.19 Confusion matrix capturing the classification results of the QMLP-2 model on the RPM and Gear spoofing attacks.

Attack	Message Type	Predicted Normal	Predicted Attack
RPM	True Normal	40221	0
	True Attack	0	9779
Gear	True Normal	41352	9
	True Attack	0	8639

message buffer on the PS to the PL, execution time and to read back the results to the software task on the PS. Note that the PS task that initiates the IDS operation is non-blocking, allowing the processor to do other tasks during the IDS execution. The latency measurement is compared against others reported in the literature in table 3.20, providing a comparison against approaches that run different models per message or per block of messages and on a variety of platforms as isolated IDS and loosely coupled accelerators. The tightly coupled IDS in the proposed ECU model achieves an end-to-end message processing latency of 0.24 ms for both models since they are executed concurrently. This is a $2.3\times$ speed up compared to MTH-IDS implemented on a dedicated IDS-ECU on a Raspberry Pi 3 device. A significant speed-up is observed compared to a loosely coupled GPU architecture performing inference (on the Nvidia A6000) as shown in table 3.21; a portion of the higher

Table 3.20 Per-message latency comparison against other state-of-the-art IDSs reported in the literature.

Models	Latency	Frames	Platform
GRU [94]	890 ms	5000 CAN frames	Jetson Xavier NX
MLIDS [31]	275 ms	per CAN frame	GTX Titan X
Rec-CNN [32]	117 ms	128 CAN frames	Jetson TX2
NovelADS [93]	128.7 ms	100 CAN frames	Jetson Nano
GIDS [91]	5.89 ms	64 CAN frames	GTX 1080
DCNN [18]	5 ms	29 CAN frames	Tesla K80
TCAN-IDS [92]	3.4 ms	64 CAN frames	Jetson AGX
MTH-IDS [23]	0.574 ms	per CAN frame	Raspberry Pi 3
QMLP-IDS (ours)	0.24 ms	per CAN frame	Zynq Ultrascale+

latency can be attributed to the PCIe overheads in small data size transactions in a per-message execution mode. In terms of raw throughput, the tightly coupled IDS-ECU can process 4166 messages per second scanning for all four attacks. This translates to a line-rate detection at 462 Kbps on a high-speed CAN network (at max data size) using the lightweight DPU. While GPU-based methods have reported better average throughput by processing a block of CAN messages at once, the detection delay incurred by the time taken to accumulate the block size of messages is ignored in such calculation. For instance, accumulating 64 CAN messages at peak data rate (1 Mbps) takes 8.3 ms, which is not factored into the reported detection latency of 5.89 ms in the case of GIDS in table 3.20.

Like the MA-QCNN model, the cold-start setup time of IDS to determine the delay from power on to the model being ready to perform IDS. In this case, it is observed that this cold-start latency is 8.9 seconds when averaged across multiple runs.

3.4.7 Power consumption, PS/PL utilisation

The power consumption of the IDS-ECU is quantified using the PYNQ-PMBus package to monitor the power rails directly and average them over 100 runs. At idle, with the ECU booted up with the Linux image, an average consumption to be 3.43 W is observed from the power rails, which rises to 3.76 W when CAN messages are passed to the IDS for execution. Considering the total power consumed by the device, this leads to the average power consumption of 0.9 mJ per inference. To model a loosely coupled GPU accelerator, the power consumption of the model is

Table 3.21 Per-message latency and measured power of the proposed model on the different platforms.

Platform	Latency (ms)	Measured Power	
		Idle (W)	Active (W)
QMLP on RTX A6000	35	22	56
QMLP on IDS-ECU	0.24	3.43	3.76

Table 3.22 Resource utilisation on the PL for the proposed CAN IDS-ECU (XCZU7EV).

Node	LUT	FF	BRAM/URAM	DSP
DPU:Core-1	27020	33889	41.5/12	110
DPU:Core-2	27018	33953	41.5/12	110
Overall	56733	72147	87/24	220
(% usage)	(24.6%)	(15.6%)	(27.8%/25%)	(12.73%)

measured on an A6000 GPU using the Nvidia 'nvidia-smi' plug-in (on a workstation machine) and observe an active power consumption of 56W (GPU only) incurring 1.96 J per inference. The IDS-ECU approach hence achieves a $15\times$ reduction in power consumption as shown in table 3.21 and a $2172\times$ better energy efficiency per inference, rendering this design a more efficient architecture for integrating IDS capabilities across ECUs. Among other reported results in the literature, the proposed approach incurs a $2.6\times$ reduction in power consumption when compared to the GRU [94] model on an Nvidia Jetson Xavier as a dedicated IDS node.

Finally, the overall hardware resource consumption of the IDS is quantified in table 3.22 and observed a peak consumption of 27.8% of resources for BRAMs and 24.6% for LUTs. This leaves enough resources ($> 75\%$ LUTs & $> 85\%$ DSP blocks) on the PL to allow other ECU functionality to be offloaded for hardware acceleration. The utilisation of the ARM cores is also quantified for managing the IDS accelerator to estimate the overhead in consolidating the IDS capability on the ECU. It was observed that a single core utilisation peaked at 40% when processing the completion interrupt from the IDS core (for each message at near line rate), while other cores remained at IDLE ($\leq 2\%$ utilisation). This could be further refined by processing the IDS outputs in hardware to flag the threats to the ECU, while also improving the detection latency.

3.5 Summary of DPU-accelerated quantised models

This section summarises the results of the three model implementations discussed in the previous section that are proposed as IDSs for automotive CAN.

Table 3.23 shows the number of parameters, the input features used & the device on which it is implemented. The models are increasingly compressed as evidenced by the reduction in the number of parameters at each iteration of the model, from QdCNN to QMLP. Also, the QMLP model uses both the CAN ID & Data field as the input features instead of only the CAN IDs as in the case of QCNN models. Table 3.25 shows the improvement in the total number of misclassifications from the QdCNN to QMLP models for each attack. A strong reduction in misclassifications is observed for the fuzzy and spoofing attacks. Figure 3.6 shows the latency improvement ($2.3\times$) compared to the state-of-the-art IDSs proposed in the research literature. Table 3.24 shows the improvement in the energy consumed per inference from the QdCNN to QMLP models ($3.3\times$ reduction) and the chosen DPU configuration for each model.

From the above results, it is observed that dense models worked very accurately for CAN data in the supervised learning scenario. The QMLP model however still does not achieve a latency of $< 128\mu\text{s}$ or 0.128ms to achieve line rate detection in CAN as discussed in section 2.2. In the next section, a highly quantised QMLP accelerator is developed that meets the above latency constraint.

Models	Parameters	Input features used	Device
QdCNN	2,757,066	CAN ID	ZCU104
MA-QCNN	608,985	CAN ID	Ultra96-V2
QMLP	53,300	CAN ID + Data Field	ZCU104

Table 3.23 Number of parameters, input features and device used for the three IDS implementations.

Models	Energy Consumption (mJ)	DPU (Mhz)
QdCNN	3.05	B1152 (600)
MA-QCNN	0.86	B1152 (300)
QMLP	0.9	B512 (600)

Table 3.24 Per-inference energy consumption and the chosen DPU configuration (clock frequency) for each IDS model.

Models	DoS	Fuzzy	RPM-Spoof	Gear-Spoof
QdCNN	5	235	527	588
MA-QCNN	13	174	-	-
QMLP	13	53	0	9

Table 3.25 Number of misclassifications for each model on all the four attacks. Each model was tested with 50000 messages for each attack.

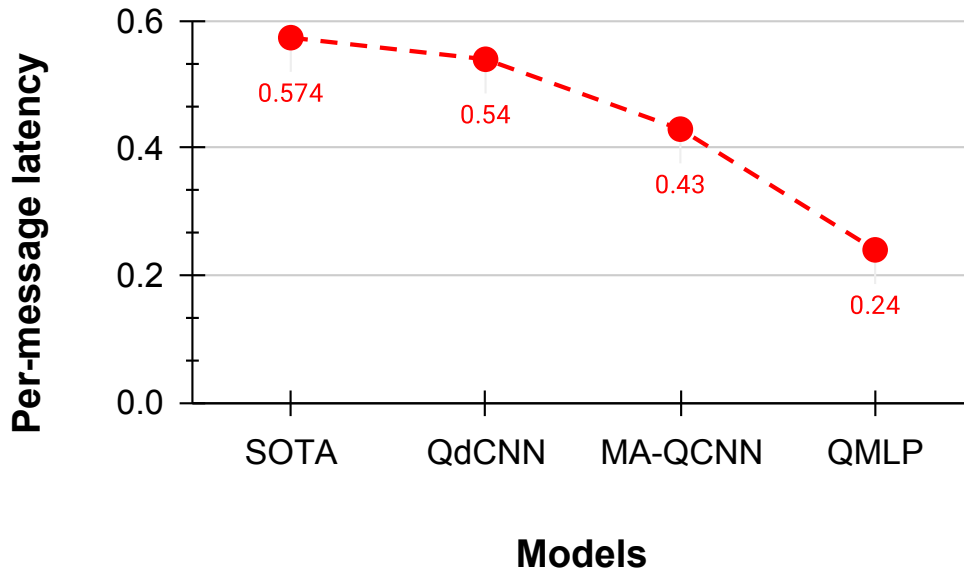


Fig. 3.6 Latency improvement observed with increased model optimisation from QdCNN to QMLP.

3.6 Custom quantised MLPs as IDSs : CQMLP

3.6.1 Model Architecture

The baseline architecture for the CQMLP is derived from the QMLP model previously discussed. Based on empirical testing, a concatenation of $n = 4$ successive messages—each comprising CAN IDs and their corresponding payloads is used as the input feature. During both training and testing, any block containing at least one attack message is labelled as an attack block. In a real-world deployment, this approach ensures that a window of four messages containing even a single malicious instance is flagged as suspicious, thereby capturing not only the attack but also

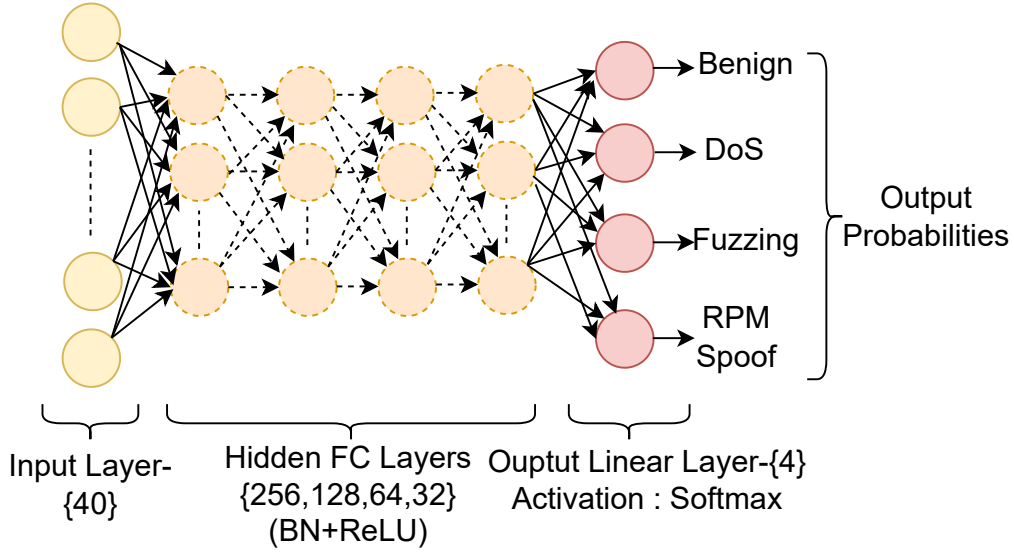


Fig. 3.7 The proposed MLP model as a multi-attack detection IDS.

the preceding and subsequent messages that may contribute to or result from the intrusion. While conservative, this strategy enhances the model's ability to learn normal operational patterns and contributes to a more balanced dataset.

The proposed MLP model comprises five Linear layers with 256, 128, 64, 32, and 4 units, respectively, as illustrated in Figure 4.2. The architecture is implemented in PyTorch using standard functions, with quantised versions of the Linear and ReLU activation layers sourced from the Brevitas training library [129]. This model differs from its predecessor by enabling the detection of DoS, fuzzing, and spoofing attacks within a single unified framework. This capability is made possible through the application of quantisation-aware training (QAT) methodology and the conservative labelling methodology, which are discussed next. The training and testing scripts for the model have been open-sourced and are available at the following repository: Quantised-MLP-Intrusion-Detection-Controller-Area-Network.

3.6.2 Design space exploration

To arrive at the optimum bit-precision for the quantised model (to improve detection latency), the MLP model was trained at 2-bit, 3-bit, and 4-bit uniform quantisation on all the attacks to determine the trade-offs in detection accuracy and resource consumption (indirectly leading to power consumption). Figures 3.8 & 3.9 show the training and validation losses when the model was trained for 1000 epochs at each quantisation level. The 2-bit model converges to an optimum detection and is

Table 3.26 Inference cost of the custom quantised models (CQMLP) inferred through the FINN library and normalised to the 4-bit model.

CQMLP Models	2-bit	3-bit	4-bit
Normalised inference cost	0.27	0.67	1

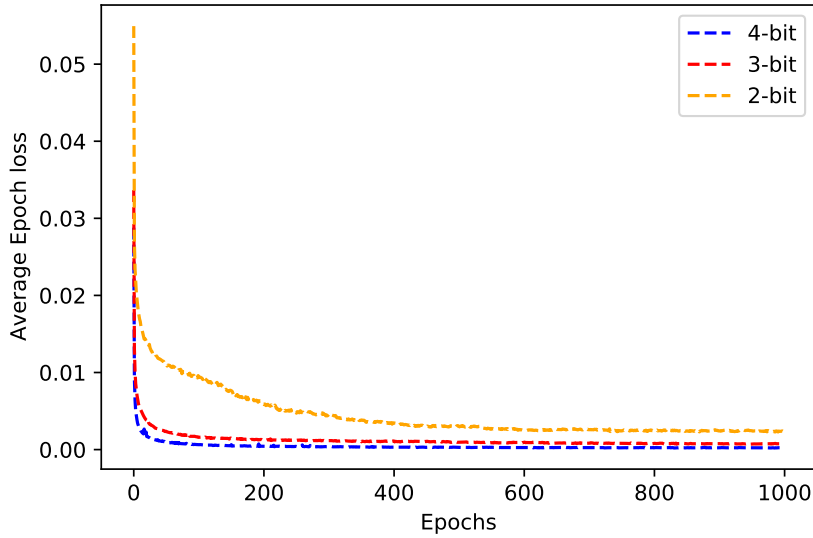


Fig. 3.8 Training loss of the model for different precision of weights and activations for all the attacks.

very close to the 3-bit & 4-bit variants (between epoch 810 and 820) in terms of validation loss. 147 misclassifications are observed out of the total 180,000 messages in the test set for the 2-bit variant. For the same test set, the misclassifications were found to be 74 & 31 for the 3-bit & 4-bit variants respectively.

To capture the inference costs of the models further, the FINN utility function (inference cost) to estimate the memory footprint and operational complexity of each model was employed. This function captures the **memory footprint** and **binary operations** cost for each version of the model; the costs are then normalised (against a baseline) and linearly combined to arrive at the normalised inference cost for each of the three quantisation levels. Table 3.26 shows these costs with the 4-bit version as the baseline. It was found that the cost of the 2-bit version to be 40% & 27% of the 3-bit & 4-bit versions respectively, without a significant drop in detection accuracy and was hence chosen as the optimal bitwidth for deploying the CQMLP.

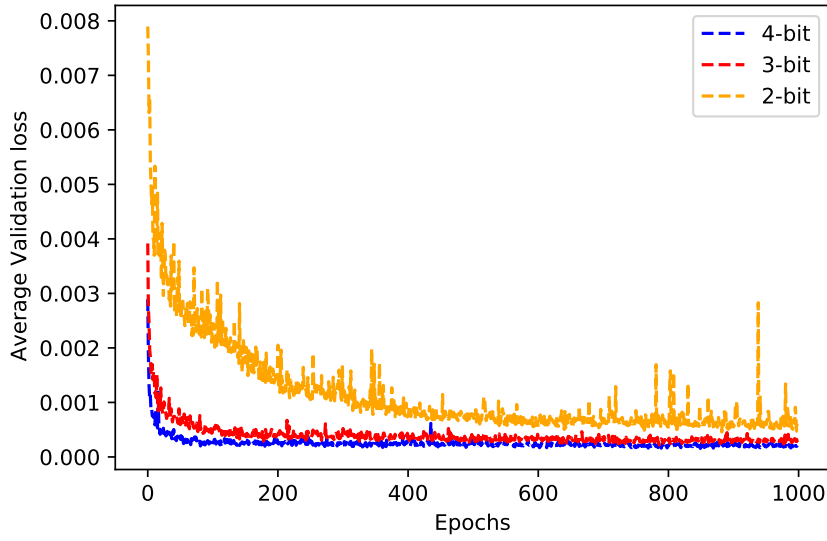


Fig. 3.9 Validation loss of the model for different precision of weights and activations for all the attacks.

3.6.3 Dataset and Training

As before, the open Car Hacking dataset was used for training the model and to test its performance, with an 85:10:5 training, validation, and testing split. Similar pre-processing steps were applied to remap the dataset contents to mimic the data received by the model when integrated into the ECU. Each message (ID and payload) is encoded into INT8 type vector, and an adjacent sequence of $n = 4$ of these encoded messages form the input feature buffer content for the IDS (block shape of $\{1, 10 \times n\}$ INT8 data).

The different bit models were trained using AMD’s QAT library, Brevitas [129]. The Adam optimiser was utilised with binary cross-entropy as the loss function. The learning rate was set to 0.0001 to allow for slower learning which aids in efficient model training [125]. Each model has 54,824 learning parameters and was trained for 1000 epochs with a batch size of 128 for the DoS, Fuzzing, and RPM-Spoof attacks. The best model in terms of validation loss was saved and exported as an ‘onnx’ graph. This graph was then fed into the FINN hardware generation flow for the CQMLP-IDS IP generation.

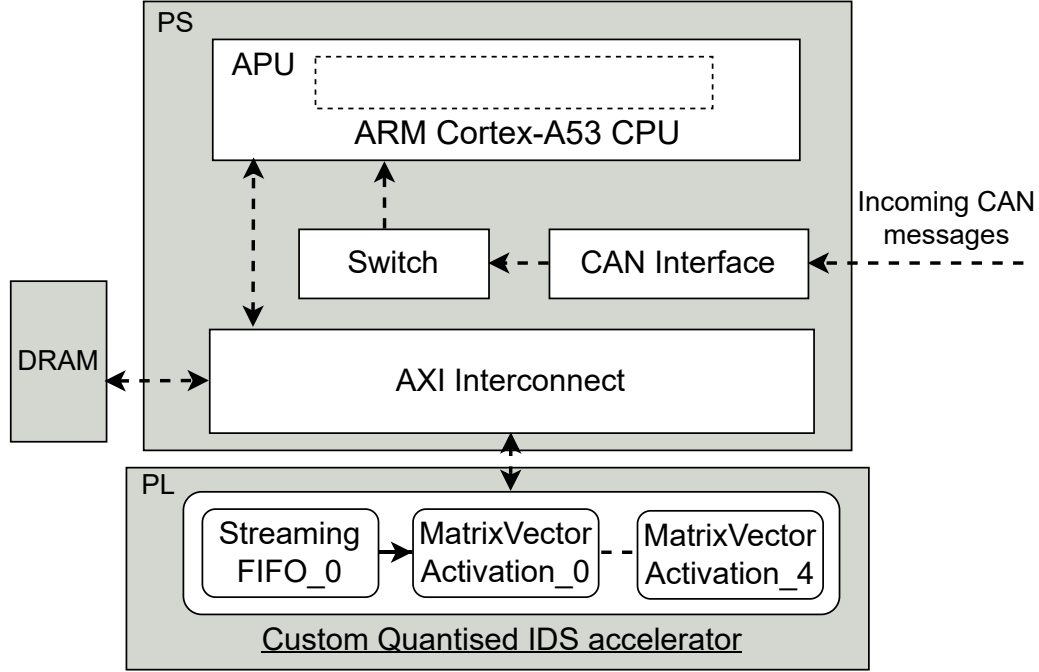


Fig. 3.10 The proposed IDS-ECU architecture. The ML model is accelerated on the PL part of the FPGA.

3.6.4 Dataflow hardware generation and integration to ECU-IDS

The FINN compiler was utilised to synthesise and generate the RTL description of the hardware implementation of the model. Starting at the 'onnx' graph, FINN applies a series of streamlining and dataflow transformations to convert the ML model into a synthesizable graph. Parallelism settings are specified to achieve the target latency (set to 100000 messages per second), matrix-vector activation unit widths (set to 80), and folding levels to generate the dataflow hardware with AXI interfaces for input and output. Subsequently, FINN invokes Vivado tools to stitch the accelerator as an AXI slave to a Zynq subsystem for the target device (XCZU7EU) and generates the final bitstream along with the software drivers for the hardware ML engine. This integration models the Zynq-based hybrid ECU architecture as previously discussed. The custom hardware accelerator is accessible from the ECU and performs IDS in complete isolation from the tasks on the ARM cores as shown in Figure 3.10. For the experiments, PYNQ runtime is used on top of a Linux kernel on the ARM cores to interact with the hardware model on the PL.

Table 3.27 Confusion matrix capturing the classification results of the CQMLP.

CQMLP	Predicted Values			
	Benign	DoS	Fuzzing	RPM-Spoof
Benign	103142	17	17	0
DoS	27	23666	0	0
Fuzzing	78	0	28003	8
RPM-Spoof	0	0	0	25042

3.6.5 Experimental Evaluation

3.6.5.1 Inference accuracy

The classification performance of the model is evaluated by identifying DoS, Fuzzy and RPM-Spoofing attacks from the Car Hacking dataset and comparing them against state-of-the-art techniques in the literature. Table 3.27 captures the classification performance of the 2-bit model in isolation across the test set as a confusion matrix. The misclassifications in the test set can be attributed to scenarios where nearly identical attack and benign message patterns are observed in a block at a given time which leads to an overall FPR of 0.03% (34 messages falsely classified as attacks out of the total 103176 benign messages).

The inference performance of the 2-bit CQMLP model is compared against the state-of-the-art IDSs and IPSs proposed in the literature: GIDS [91], DCNN [18], MLIDS [31], HyDL-IDS [130] NovelADS [93], TCAN-IDS [92], IForest [95], MTH-IDS [23], GRU [94], G-IDCS [131], TAN-IDS [132] and Rec-CNN [32] which are captured in table 3.28, comparing them in terms of inference precision, recall, F1 score and false negative rate. For the DoS attack the 2-bit CQMLP performs equally among the proposed IDSs in terms of F1 score. In the case of fuzzing attack, the 2-bit CQMLP model performs identically to DCNN [18] while performing better than iForest [95], TCAN-IDS [92], and GRU [94] by 2.3%, 0.6% and 0.6% respectively in terms of F1 score. The 2-bit CQMLP also achieves almost perfect classification for the RPM-spoofing attack and performs better than or equal to the other methods proposed in the literature. The previous QMLP model using two 8-bit variants achieved 99.96%, 99.76% & 100% F1 scores for the DoS, Fuzzing, and RPM spoofing attacks, when deployed as two concurrent accelerators for detecting them simultaneously [133]. In contrast, the 2-bit CQMLP variant achieves almost identical F1 scores (99.90%, 99.81% & 99.98% respectively) for the same attacks (from the same dataset) while performing multi-class classification using a single

Table 3.28 Accuracy metric comparison (%) of the CQMLP accelerator against the reported literature on the DoS, Fuzzing and RPM-Spoof attacks.

Attack	Model	Precision	Recall	F1	FNR
DoS	GIDS [91]	-	99.9	-	-
	DCNN [18]	100	99.89	99.95	0.13
	MLIDS [31]	99.9	100	99.9	-
	G-IDCS [131]	99.81	98.86	99.33	-
	TAN-IDS [132]	100	100	100	-
	HyDL-IDS [130]	100	100	100	0
	NovelADS [93]	99.97	99.91	99.94	-
	TCAN-IDS [92]	100	99.97	99.98	-
	iForest [95]	-	-	-	-
	GRU [94]	99.93	99.91	99.92	-
	CQMLP-IDS	99.92	99.88	99.90	0.11
Fuzzing	GIDS [91]	-	98.7	-	-
	DCNN [18]	99.95	99.65	99.80	0.5
	MLIDS [31]	99.9	99.9	99.9	-
	G-IDCS [131]	99.71	99	99.35	-
	TAN-IDS [132]	99.99	99.99	99.99	-
	HyDL-IDS [130]	99.98	99.88	99.93	-
	NovelADS [93]	99.99	100	100	-
	TCAN-IDS [92]	99.96	99.89	99.22	-
	iForest [95]	95.07	99.93	97.44	-
	GRU [94]	99.32	99.13	99.22	-
	CQMLP-IDS	99.93	99.69	99.81	0.27
RPM-Spoof	GIDS [91]	-	99.6	-	-
	DCNN [18]	99.99	99.94	99.96	0.05
	MLIDS [31]	100	100	100	-
	G-IDCS [131]	99.85	98.69	99.27	-
	TAN-IDS [132]	99.99	99.93	99.96	-
	HyDL-IDS [130]	100	100	100	0
	NovelADS [93]	99.9	99.9	99.9	-
	TCAN-IDS [92]	99.9	99.9	99.9	-
	iForest [95]	98.9	100	99.4	-
	CQMLP-IDS	99.96	100	99.98	0

inference model. By adopting QAT training and conservative block-based labelling, the highly quantised model can match the inference performance of the 8-bit QMLP variant discussed in the previous section.

Table 3.29 Per-message latency comparison against other state-of-the-art IDSs reported in the literature.

Models	Latency	Frames	Platform
GRU [94]	890 ms	5000 CAN frames	Jetson Xavier NX
MLIDS [31]	275 ms	per CAN frame	GTX Titan X
Rec-CNN [32]	117 ms	128 CAN frames	Jetson TX2
NovelADS [93]	128.7 ms	100 CAN frames	Jetson Nano
GIDS [91]	5.89 ms	64 CAN frames	GTX 1080
TAN-IDS [132]	11.6 ms	128 CAN frames	-
DCNN [18]	5 ms	29 CAN frames	Tesla K80
TCAN-IDS [92]	3.4 ms	64 CAN frames	Jetson AGX
MTH-IDS [23]	0.574 ms	per CAN frame	Raspberry Pi 3
QMLP-IDS(3.4.1)	0.24 ms	per CAN frame	Zynq Ultrascale+
CQMLP-IDS	0.11 ms	per CAN frame	Zynq Ultrascale+

3.6.5.2 Inference latency

The per-message processing latency of the model is quantified starting from the arrival of the CAN message at the interface to determine the detection delay incurred by the approach. Table 3.29 compares the result against other approaches in the literature, which utilise different platforms (GPUs, Jetson edge accelerators, Raspberry Pi) and approaches (block of CAN messages v/s individual messages). It should be noted that the latency metric in the case of block-based IDS does not consider the delay in acquiring the number of CAN messages required, and hence could result in potentially larger delays in detection of the onset of an attack as mentioned in the previous sections. The tightly integrated CQMLP-IDS achieves 0.11 ms per CAN frame, which is a $2.18\times$ improvement over the dedicated line-rate QMLP-IDS ECU proposed in the previous subsection. It is also observed that the inference latency of the (fp32) model on the Jetson Nano (modeling a dedicated IDS ECU) is 1.2 ms when averaged over 10000 runs. This is $10.9\times$ slower than the proposed CQMLP-IDS on the integrated IDS-ECU on the Zynq Ultrascale platform. In terms of raw throughput, the dataflow implementation of CQMLP coupled to the ECU can process over 9090 messages per second at the highest payload capacity, achieving near-line-rate detection on high-speed critical CAN networks.

Table 3.30 Resource utilisation breakdown of the proposed CQMLP (XCZU7EV).

Node	LUT	FF	BRAM/URAM	DSP
2-bit-CQMLP-IDS	3999	4524	4/0	0
% Usage	1.74%	0.98%	1.28%/ 0%	0%

3.6.5.3 Power consumption, Resource Utilisation (PS/PL)

The active power consumption of the dataflow accelerator is also quantified to determine the average energy consumption per inference and the hardware resources incurred by the dataflow design on the XCZU7EV device. It is observed that the model consumed 2.15 W when measured directly from the device’s power rails (using the PYNQ-PMBus package) while performing inference and other tasks on the ECU (with Linux OS), thus consuming 0.23 mJ of energy per inference. This is a $3.9\times$ improvement compared to QMLP-IDS which reports a per-inference energy consumption of 0.9 mJ. In comparison, the observed per-inference energy consumption of the (fp32) MLP model on the Jetson Nano is 2.76 mJ when averaged over 10000 inference runs which is $12\times$ more than the proposed CQMLP-IDS on the tightly integrated IDS-ECU on a hybrid FPGA device. Among other reported results in the literature, the proposed approach improves the active power consumption by $4.6\times$ when compared to the GRU [94] model on an Nvidia Jetson Xavier as a dedicated IDS node. In terms of resources, the dataflow CQMLP model consumes less than 2% of resources on the device, leaving behind enough resources to integrate dedicated accelerators for the ECU function.

3.7 Summary

In this chapter, incrementally optimised quantised CNN and MLP models are presented as IDSs for controller area network. The results in the chapter demonstrate that the 2-bit quantised model achieved line-rate detection speeds, with a per-message processing latency of just 0.11 ms ($< 0.128\text{ms}$ which is the interframe arrival time when the CAN bus operates at 1Mbps as discussed in section 2.2).

To maintain high accuracy though, this model required processing a minimum block of four messages and employed a conservative labeling technique coupled with QAT flows. Combining the labelling techniques and optimisations in the dataflow generation, it successfully detected multiple attack vectors using a single model. It

was observed that this was the maximum performance that could be extracted from feed-forward neural networks.

This chapter primarily focused on developing supervised learning-based quantised IDS models. However, the evolving nature of attack vectors necessitates frequent updates to these models, resulting in increased maintenance overhead. In contrast, unsupervised learning models—trained solely on benign messages and capable of generalising without labelled attack data—offer promise in detecting previously unseen or unknown attack vectors. Therefore, the next chapter explores the use of unsupervised learning approaches for zero-day attack detection and their deployment in automotive CAN.

3.8 Related Publications

The following publications are related to this chapter:

- S. Khandelwal, E. Wadhwa and S. Shreejith, "Deep Learning-based Embedded Intrusion Detection System for Automotive CAN," 2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (**ASAP**), Gothenburg, Sweden, 2022, pp. 88-92
- S. Khandelwal and S. Shreejith, "A Lightweight Multi-Attack CAN Intrusion Detection System on Hybrid FPGAs," 2022 32nd International Conference on Field-Programmable Logic and Applications (**FPL**), Belfast, United Kingdom, 2022, pp. 425-429
- S. Khandelwal and S. Shreejith, "A Lightweight FPGA-based IDS-ECU Architecture for Automotive CAN," 2022 International Conference on Field-Programmable Technology (**ICFPT**), Hong Kong, 2022, pp. 1-9
- S. Khandelwal, A. Walsh and S. Shreejith, "Quantised Neural Network Accelerators for Low-Power IDS in Automotive Networks," Design, Automation & Test in Europe Conference & Exhibition (**DATE**), Antwerp, Belgium, 2023, pp. 1-2
- S. Khandelwal and S. Shreejith, "Exploring Highly Quantised Neural Networks for Intrusion Detection in Automotive CAN," 2023 33rd International Conference on Field-Programmable Logic and Applications (**FPL**), Gothenburg, Sweden, 2023, pp. 235-241

Chapter 4

Zero-Day Attack Detection on CAN networks

4.1 Introduction & Contribution

The supervised learning-based intrusion detection approaches discussed in the previous chapter and the literature learn the signature of attack patterns from the dataset and classify any similar messages as potentially malicious during testing. Though supervised learning models can be extremely good at detecting known attacks on which they are trained, they lack the capability of detecting new attack vectors referred to as **zero-day attacks** [133]. Figure 4.1 illustrates how an intelligent attacker can evade supervised learning-based IDS models, thereby exposing vehicular networks to potential vulnerabilities. IDS approaches that rely on unsupervised learning models have been shown to be successful in detecting zero-day (novel) attacks while simultaneously classifying known attacks successfully [23, 91]. With increasing connectivity and automation in vehicles, the potential for novel attack vectors is also likely to increase, making the detection of such attacks along with known attack vectors a priority [134]. Despite their promising performance, their deployment and integration face similar challenges to supervised models – from multi-standard networks to different message sequences across vehicles and submodels and restrictions due to platform and system capabilities.

In this chapter, a convolutional autoencoder model (CAE) is developed as a zero-day IDS function, which is integrated through a hardware-efficient ML engine to meet real-time performance while offering complete isolation to the ECU software function. The CAE models are 8-bit-quantised using AMD/Xilinx’s Vitis-AI framework and

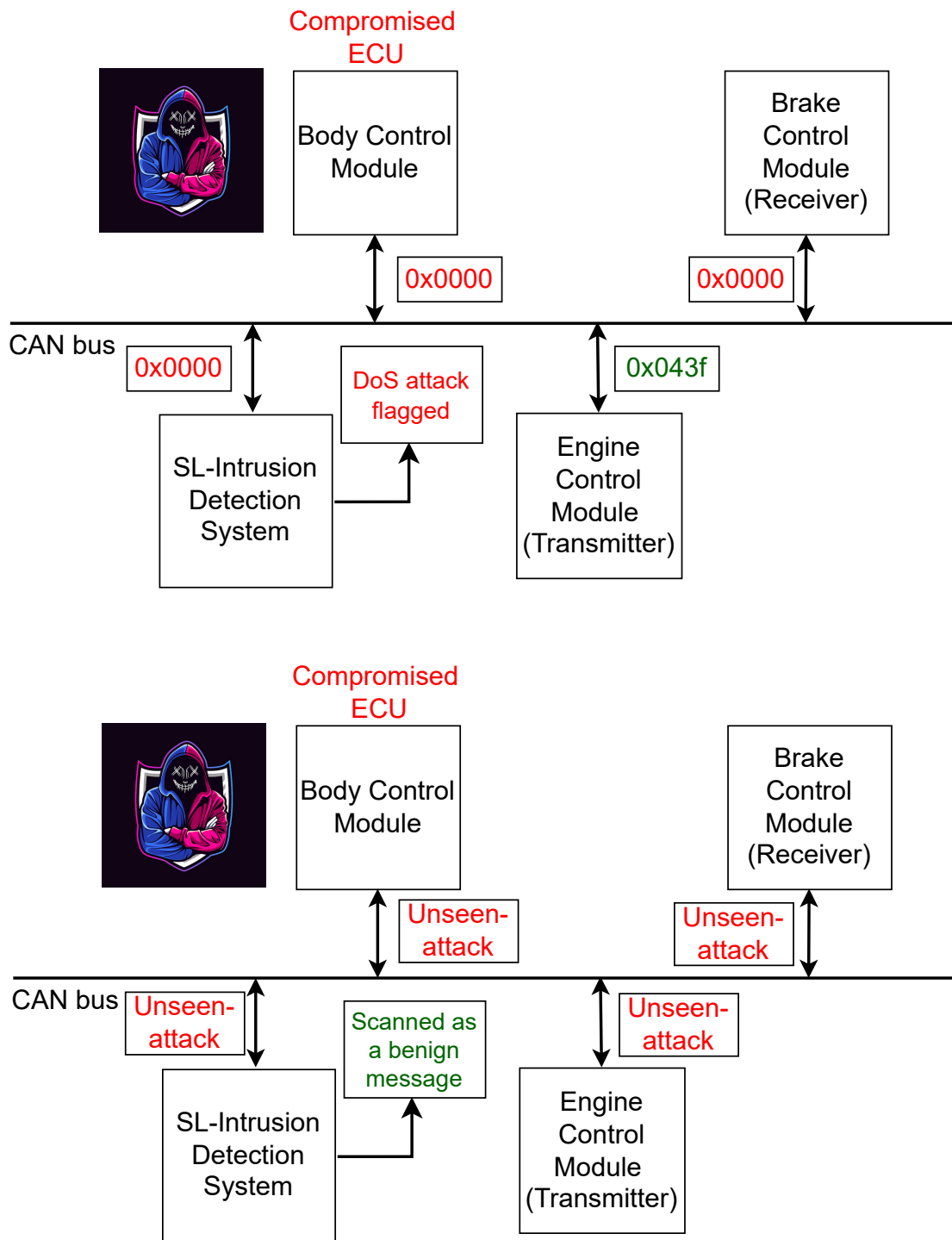


Fig. 4.1 An illustration of a zero-day attack launched through a compromised ECU left undetected by an IDS. The top figure shows a supervised learning (SL) based IDS flagging the known DoS attack on the CAN bus, while in the bottom figure, the IDS is unable to flag the attack on the CAN bus as the compromised ECU uses zero-day attack techniques.

are deployed as an AXI slave peripheral of the ECU for enabling IDS capabilities at the ECU-level [111]. As before, this integration allows the software tasks on the ECU to invoke and fully control the operation of the IDS accelerator through APIs, similar to compute offloads enabled by AUTOSAR abstractions as discussed in the previous chapter. The key contributions of this chapter are as follows:

- An unsupervised learning model (quantised convolutional autoencoder) based-IDS was developed for automotive CAN that models ‘normal’ CAN messages to act as a generalised anomaly detector.
- Replicated the tightly coupled IDS deployment architecture for integrating the custom ML-based accelerator for offloading IDS tasks in full isolation from the ECU.
- The performance and energy savings of the proposed unsupervised models and their integration are quantified on the two open CAN datasets. The results show that the proposed IDSs achieve significant improvements in terms of per-message processing latency and power consumption against the state-of-the-art unsupervised learning-based IDSs proposed in the research literature.

To validate the generalisable nature, the model is evaluated using the openly available CAR Hacking dataset and Survival Analysis dataset (referred to as CH and SA in this chapter) for training and validation of the models on normal CAN messages and testing it across multiple unseen attack vectors captured from actual vehicles [18, 96]. The training and testing scripts for the model have been open-sourced and are available at the following repository: Unsupervised-Learning-Autoencoder-Anomaly-Detection.

The remainder of this chapter is organised as follows: Section 4.2 discusses relevant literature on unsupervised learning-based models as IDSs; Section 4.3 discusses the architecture choice of autoencoders as efficient anomaly detectors; Section 4.4 presents the autoencoder design flow, architecture choice and its integration architecture; Section 4.5 presents the experimental results and compares them against SOTA IDSs presented in the literature and the chapter is concluded in Section 4.6.

4.2 Unsupervised learning models as IDSs

Multiple supervised learning-based IDSs have been proposed and shown to be highly successful in detecting various attack vectors [18, 31, 94, 133]. Unsupervised learning

models on the other hand have also been shown to generalise fairly well on benign datasets to detect unseen attacks including zero-day. To determine their efficiency, they are tested against an attack dataset to detect attacks like DoS, fuzzing, and spoofing [23, 91, 93, 95]. Table 4.1 shows the state-of-the-art IDSs proposed in the literature along with the learning strategy and the input features used to train their models. Here, supervised and unsupervised learning are denoted as SL & UL respectively.

In [91], the authors propose a GAN-based UL IDS and achieve an average accuracy of 97.5% for the unseen (DoS, fuzzing, and spoofing) attacks. In [95], the authors use an isolation forest unsupervised anomaly detection algorithm as an intrusion prevention system (IPS) to detect fuzzing and spoofing (RPM & Gear) attacks and mark the message as an error preventing its propagation to other ECUs; however, this can cause multiple messages to be dropped from the bus in case of false positives or DoS attacks. In [135], the authors proposed the use of stacked autoencoders and GMMs as IDSs trained only on benign messages. Their technique however results in accuracy $< 80\%$, making their model less suited for deployment. In [23], the authors proposed a stacked anomaly-based IDS for detecting zero-day attacks and reported a slightly lower average F1 score of 96.3% on the attacks included in the CH dataset. In [93], the authors propose an unsupervised learning-based CNN+LSTM autoencoder architecture for incoming message reconstruction and classifying them as benign and attacks using thresholding techniques. They report high accuracy of 99.9% for all attacks but their high message processing latency makes them unsuitable for real-time detection. In [136], authors propose the use of uses an extreme gradient boosting machine learning algorithm to categorise unexpected occurrences in the CAN data payload. They report accuracies $> 99\%$ on the SA attack dataset. In [137], authors propose LSTM-based IDS to detect and mitigate the CAN bus network attacks and achieve accuracies $> 99.9\%$ on the CH and SA attack datasets. Many successful approaches presented in the literature utilise variations of autoencoders to build UL-based IDSs and hence, these models have been explored in this chapter.

The key challenge of these models stems from their increased complexity restricting their deployment as a practical in-vehicle solution. Most approaches rely on high-performance GPUs to meet the inference deadlines [18, 31, 91–94], while others rely on dedicating full ECUs for IDS [23, 95]; both approaches incur additional overheads in energy budget, integration, and weight, making them less suited for integration among different network segments. Additionally, most (if not all) of the presented works in the literature utilise compute-intensive floating-point representations to achieve

Table 4.1 Input features and training methods used by state-of-the-art IDSs & IPSs proposed in the research literature.

Models	Input Features	Learning
GIDS [91]	CAN ID	UL
DCNN [18]	CAN ID	SL
Rec-CNN [32]	CAN ID	SL
iForest [95]	Data Field	UL
MLIDS [31]	CAN ID + Data Field	SL
NovelADS [93]	CAN ID + Data Field	UL
TCAN-IDS [92]	CAN ID + Data Field	SL
MTH-IDS [23]	CAN ID + Data Field	UL
HyDL-IDS [130]	CAN ID + Data Field + DLC	SL
GRU [94]	CAN ID + Data Field + DLC	SL
XGBoost [136]	CAN ID + Data Field	SL
Random Forest [138]	CAN ID + Data Field	SL
LSTM [137]	CAN ID + Data Field + DLC	SL
QCAE-IDS (proposed)	CAN ID	UL

the claimed inference accuracy. It should be noted that the floating-point operations require explicit support from the ECU architecture for optimal performance and are otherwise relying on software emulation which can create massive overheads.

4.3 Autoencoders as anomaly detectors

Autoencoders are a class of neural networks designed for the **faithful reconstruction** of input features they are trained on, following an **encoder-decoder** architecture. These networks learn a compressed representation of the input data, capturing its most essential features in a lower-dimensional latent space.

During training, the encoder block extracts meaningful patterns from high-dimensional input representations, transforming them into a compact form stored in the **latent space** (also referred to as the **bottleneck**). This process ensures that only the most salient information is retained while redundant or less important features are discarded. The encoder network can be mathematically represented as follows:

$$z = \sigma(Wx + b) \quad (4.1)$$

where:

- σ represents the activation function,
- x is the input feature vector,
- W and b denote the weights and biases of the encoder network,
- z is the learned latent representation.

The size of the latent space is a critical design choice—a smaller latent space forces the model to generalise better, reducing overfitting, while a larger latent space may capture unnecessary details.

Once the input data is encoded into this latent representation, the decoder block reconstructs the original input using the encoded information. The decoder network can be expressed as:

$$\tilde{x} = \tilde{\sigma}(\tilde{W}z + \tilde{b}) \quad (4.2)$$

where:

- $\tilde{\sigma}$ is the decoder activation function,
- $\tilde{W}$ and $\tilde{b}$ represent the decoder weights and biases,
- $\tilde{x}$ is the reconstructed output.

The training process minimises the reconstruction error, often measured using a loss function such as **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)** or **Binary Cross Entropy (BCE)** depending on the output. Training is considered successful when the autoencoder learns to reconstruct inputs with minimal loss and preserves the key input information. Autoencoders have diverse applications, including:

- **Dimensionality reduction** – extracting lower-dimensional representations of high-dimensional data [139].
- **Image denoising** – removing noise while preserving essential image details [140].
- **Data generation** – creating new samples similar to the training distribution [141].
- **Anomaly detection** – identifying deviations from normal patterns in data [142].

In anomaly detection, autoencoders leverage their reconstruction ability to differentiate between normal and anomalous data. Since they are trained only on normal patterns, they struggle to reconstruct inputs that significantly deviate from the training distribution. This results in a higher reconstruction loss for unseen, unpredictable, or out-of-sequence data points.

To systematically identify anomalies, a thresholding technique is applied:

1. A threshold is determined based on the reconstruction loss of normal data.
2. During testing, any data point that produces a reconstruction loss exceeding this threshold is flagged as an anomalous instance.

This approach is particularly useful in network intrusion detection, medical diagnostics, and fault detection, where identifying deviations from normal patterns is crucial. In the next section, the system architecture (model exploration/development, training & integration architecture) of the UL-based IDS model is presented.

4.4 System architecture

4.4.1 CAE model as unsupervised IDS

To determine the best-unsupervised ML model as IDSs, various ML architectures were profiled, including dense autoencoders, convolutional autoencoders, and hybrid models. The aim here again was to identify a model architecture that balances high classification accuracy (generalisable across multiple attack datasets) with low computational complexity.

Among the candidate architectures, it was observed that CAEs were particularly effective at detecting attacks when trained using a window of CAN-IDs as the input feature. This input feature allowed the model to leverage the spatial structure of the input CAN messages, improving attack detection performance. Two versions of the CAE model are presented in this chapter with the same architecture but different hyperparameters for the CH and SA attack datasets respectively [18, 96].

The architecture of the autoencoder is based on an encoder-decoder framework. The encoder consists of two **Conv2D** layers with **ReLU** activation, and max-pooling layers are inserted between the convolutions to progressively reduce the feature dimensions. The decoder is composed of two **Conv2DTranspose** (Deconvolution) layers, also using **ReLU** activations, to reconstruct the input from the learned latent space. The final layer of the decoder consists of a **Conv2D** layer with a single filter

and `sigmoid` activation, producing the binarised reconstructed message. This model architecture is shown in Figure 4.2. The model was implemented in Tensorflow, using standard TensorFlow functions and node definitions.

As the input feature, a sequence of $n = 100$ CAN-IDs is chosen in binary format while experimenting with the CH and SA datasets, capturing the sequence of messages exchanged on a normal CAN network. At a bus utilisation of 100%, this corresponds to a time window of 1 ms (with the peak rate of 10,000 messages per second) on a high-speed CAN network.

For both datasets, during the exploration, it was observed that using smaller blocks (e.g., $n = 10$ or $n = 50$) limited the model’s ability to learn intrinsic features, reducing the classifier’s efficiency. On the other hand, increasing the window size (i.e., block size $n > 100$) resulted in much higher worst-case detection latency, as more time was required to accumulate the required number of messages for attack detection and hence $n = 100$ was chosen as the block size for this dataset.

In addition to using only CAN IDs as the input feature used for training the models, CAE models trained using windows of message payloads were also explored to enhance the system’s ability to detect zero-day attacks. By leveraging both message IDs and payloads, the aim was to improve the robustness and generalisation of the intrusion detection mechanism. During these experiments, the CAE model demonstrated strong performance in detecting fuzzing attacks, achieving $> 99\%$ accuracy when using a relatively small input window of $n = 100$ consecutive message payloads. However, the model’s performance was less effective against more subtle, targeted spoofing attacks that closely mimic legitimate in-vehicle communication patterns. For instance, in the CH dataset, attacks such as RPM and Gear spoofing—which generate payloads statistically similar to normal ECU traffic—proved more difficult for the model to distinguish from benign behavior. This proved to be a key challenge in CAE-based zero-day intrusion detection: while payload-based models can successfully detect high-entropy or abnormal inputs like fuzzing, they struggled to flag sophisticated attacks that exhibit semantic consistency with normal ECU communication.

4.4.2 Design space exploration for CAE

To determine the optimal configuration of the CAE, different design configurations with different layers and filter sizing were explored, iterating through multiple training and validation steps. For the `Conv2D` and `Conv2DTranspose` layers different filter sizes $\{128, 64, 32\}$ both in the encoder and decoder were explored. It was found that

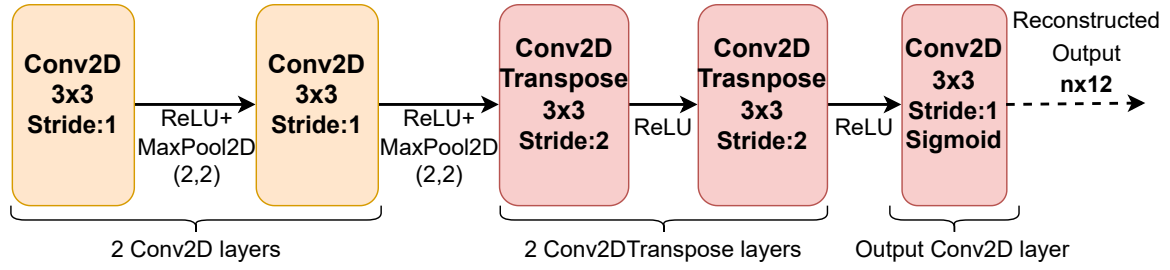


Fig. 4.2 The proposed convolutional autoencoder model architecture as a zero-day attack detection IDS. ‘n’ denotes the window size of the input feature used for attack detection. The proposed model uses $n = 100$ in this chapter.

the configuration $\{128, 64\}$ & $\{64, 128\}$ gave the highest classification performance for the CH dataset while keeping the computational complexity of the model low. For the SA dataset, it was found that the configuration $\{64, 32\}$ & $\{32, 64\}$ for the filter sizes in the encoder and decoder gave the highest classification performance. Due to the significantly lower volume of normal message data (less than $9 \times$) compared to the CH dataset, the smaller model converges more quickly and prevents overfitting when employing the larger model architecture.

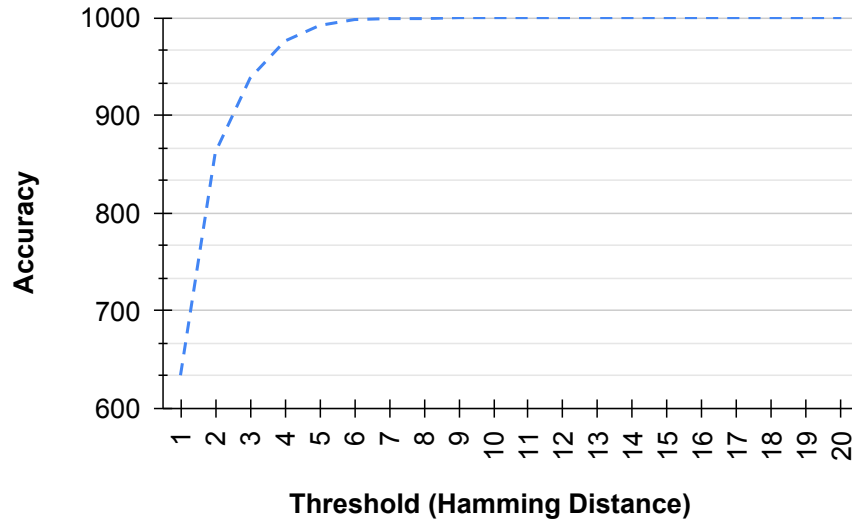
To classify the reconstructed messages as benign or attack the **hamming distance** metric was chosen to compute the distance (threshold) between the original and predicted binary vectors. When the networks trained on normal ECU communication messages encounter attack messages in the input stream, the reconstructed message results in a much larger hamming distance (greater than the classification threshold) as the model has not seen these messages during the training phase. To arrive at the optimal classification threshold for the two datasets, the accuracy of the IDS on the benign dataset version of the two datasets is evaluated by varying the threshold from 0 to 100. For the CH dataset, it was found that for threshold values from 0 to 9, the number of false positives decreases gradually, while for values of 10 and above, zero false positives were generated for classifying normal messages, leading to 10 as the optimal classification threshold. For the SA dataset, it was found that for threshold values from 0 to 67, the number of false positives decreases gradually, while for values of 68 and above, zero false positives were generated for classifying normal messages, leading to 68 as the optimal classification threshold for the model trained on this dataset. Figure 4.3 shows the threshold identification graphs for the CH and SA datasets. Figure 4.4 shows the attack detection graphs based on the above thresholds. Validation of both the models on the attack datasets showed that

the chosen thresholds for the models achieve very good classification results across all attack classes, detailed results of which are reported in section 4.5.

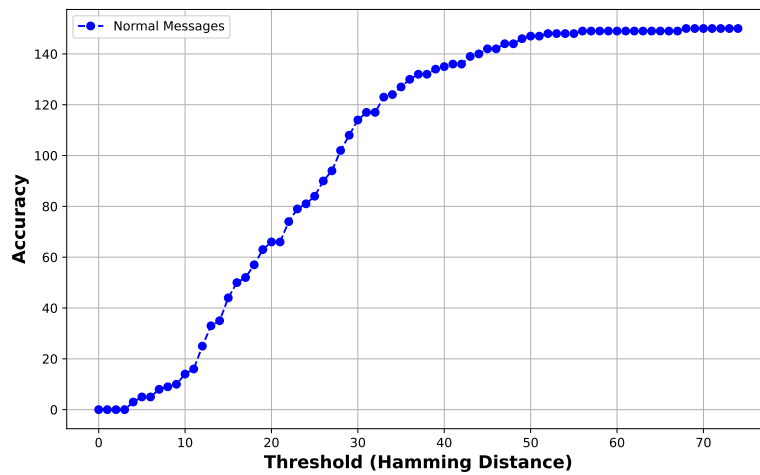
With the parameter configurations and architecture for the proposed CAE’s architecture determined through the design space exploration, the weights, biases, and activations are 8-bit quantised using AMD/Xilinx’s Vitis-AI framework [111] to generate the quantised versions of the CAE models (referred to from hereon as QCAE-CH and QCAE-SA). Note, the threshold post-quantisation for QCAE-CH model remains the same, while for the QCAE-SA model, it shifts from 67 to 73, which is then used to verify the functional correctness of that model. For both models, different hardware accelerator models were explored using different configurations of Vitis-AI’s DPU IP core to deploy the QCAE IDS [112]. Based on the resources available on the target Ultrascale+ device (XCZU7EV), three different DPU configurations (B512, B1152, B4096) were evaluated to compare the latency and energy consumption of the models, with B4096 DPU providing the best tradeoff for the QCAE-CH model while the B512 DPU providing the best tradeoff for the QCAE-SA model (results in section 4.5). For the inference results and comparisons, these DPU configurations were chosen as the accelerator.

4.4.3 Dataset and Training

The open CH dataset and the SA dataset were used for training the models and to test their performance [96, 128]. The datasets provide a labelled set of normal (benign) and attack messages which were captured via the OBD-II port in an actual vehicle, with attack messages injected in real-time. The datasets consisting of attack-free set of messages are used to train the CAE models helping it generalise on the normal flow of CAN traffic. The attack datasets—comprising DoS, Fuzzing, and Spoofing message injections in the CH dataset, and Flooding, Fuzzing, and Malfunction attacks in the SA dataset—enable us to evaluate the IDS’s detection accuracy against these previously unseen threats. The normal message set of both the datasets were split as 75:15:10 for training, validation, and testing respectively, allocating the large section to training and optimisation of the quantised network. From the four attack datasets used for testing in the CH dataset, 200,000 messages were extracted in sequence from each of them, resulting in 2000 blocks $n = 100$ of messages for testing. If the block comprises one or more attack messages the entire block is labelled as an attack block. Similarly, for the SA dataset, 100,000 messages were extracted in sequence from each of the attack sets, resulting in 1000 blocks $n = 100$ for testing.



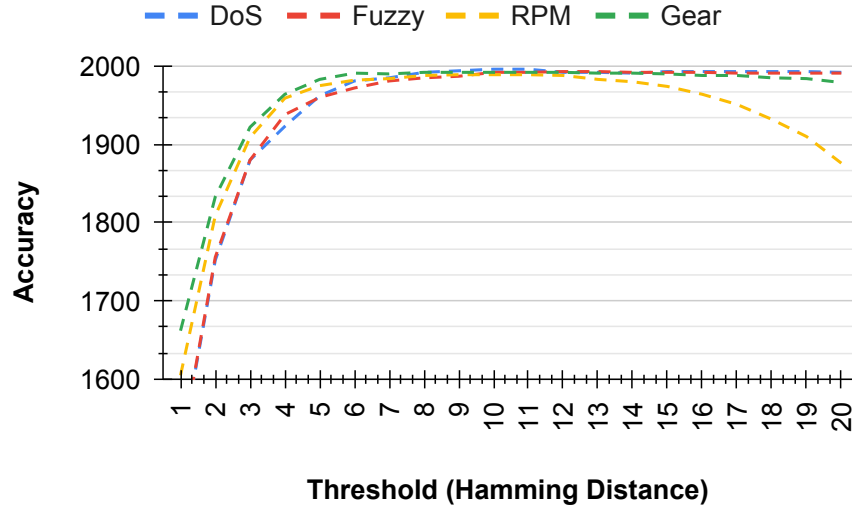
(a) Hamming distances of the reconstructed messages on the benign (attack-free) set of CH dataset. 1000 blocks of CAN messages were used in testing for this set.



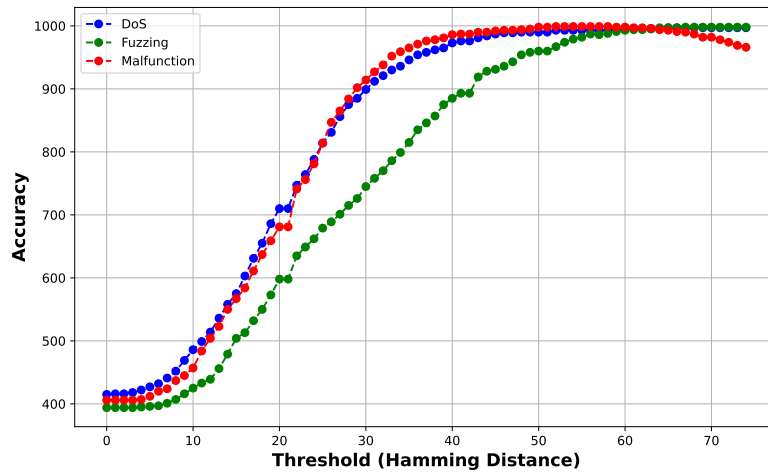
(b) Hamming distances of the reconstructed messages on the benign (attack-free) set of the SA dataset. 150 blocks of CAN messages were used in testing for this set.

Fig. 4.3 The figure illustrates the performance of the proposed models on the attack-free portions of the datasets, which are subsequently used to determine the attack detection thresholds.

The performance of the model on the validation set during training ensures that it is not overfitting on the benign message dataset. As with previous cases, pre-processed (as blocks of 100 messages) were applied to the dataset before training to mimic the CAN data format the model will obtain as its input when integrated into the



(a) Hamming distances of the reconstructed messages on the DoS, fuzzing, and spoofing attacks of the CH dataset. 2000 blocks of CAN messages were used in testing for each attack set.



(b) Hamming distances of the reconstructed messages on the Flooding, fuzzing, and malfunction attack datasets from the SA dataset. 1000 blocks of CAN messages were used in testing for each attack set.

Fig. 4.4 The figure illustrates the accuracy of the proposed models in detecting unseen attacks from the CH and SA datasets, using thresholds derived from testing on their respective benign subsets.

ECU. With 12-bit CAN IDs used in the dataset, the binarised input block has the shape $\{100, 12\}$ which is organised as the input FIFO/buffer when integrated with the PS. These blocks are then fed into the model as input for training and testing respectively.

The models were trained using the standard TensorFlow framework. The **Adam** optimiser was used with **binary cross entropy** as the loss function during training. The BCELoss function is given by:

$$Loss = L = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (4.3)$$

The learning rate for training the models was set to 0.001 to allow for slower learning which aids in reducing loss of accuracy when quantising the pre-trained model [125]. The larger version of the model comprises of 187,079 learning parameters while the smaller version comprises of 47,425 parameters. Each of the models was trained for 500 epochs with a batch size of 64 on the normal (attack-free) dataset. The best model in terms of validation loss was saved and exported as an 'h5' file. The weights file for the models is then fed into the Vitis-AI toolchain for quantisation and post-quantisation optimisations for final deployment on the DPU IP.

4.4.4 Integration Architecture

The integration architecture of these models closely mirrors that of the models presented in Chapter 3. However, since this model processes a significantly larger block of CAN messages (in contrast to the smaller blocks of only four messages used in Chapter 3), ping-pong buffering was incorporated to accumulate incoming CAN messages. This design enables a clear overlap between the execution of the IDS and the accumulation of the subsequent message block, as illustrated in Figure 4.5. The accuracy and deployment results for the QCAE models are discussed in the next section.

4.5 Experimental Results

For training, the floating point variant of the unsupervised CAE models used an Nvidia A6000 GPU for accelerating the training and validation, prior to quantisation through Vitis-AI. The Zynq Ultrascale+ ZCU104 platform was chosen as the deployment target to quantify resource overheads and run-time energy consumption. A standard Linux kernel with petalinux tools and VART interfaces enabled is used as the boot configuration for the ARM cores. The VART APIs are used to program the model ('xmodel' binary) onto the DPUs (in PL) at startup and to trigger the execution of the model at runtime. The A53 cores on the PS are configured to run

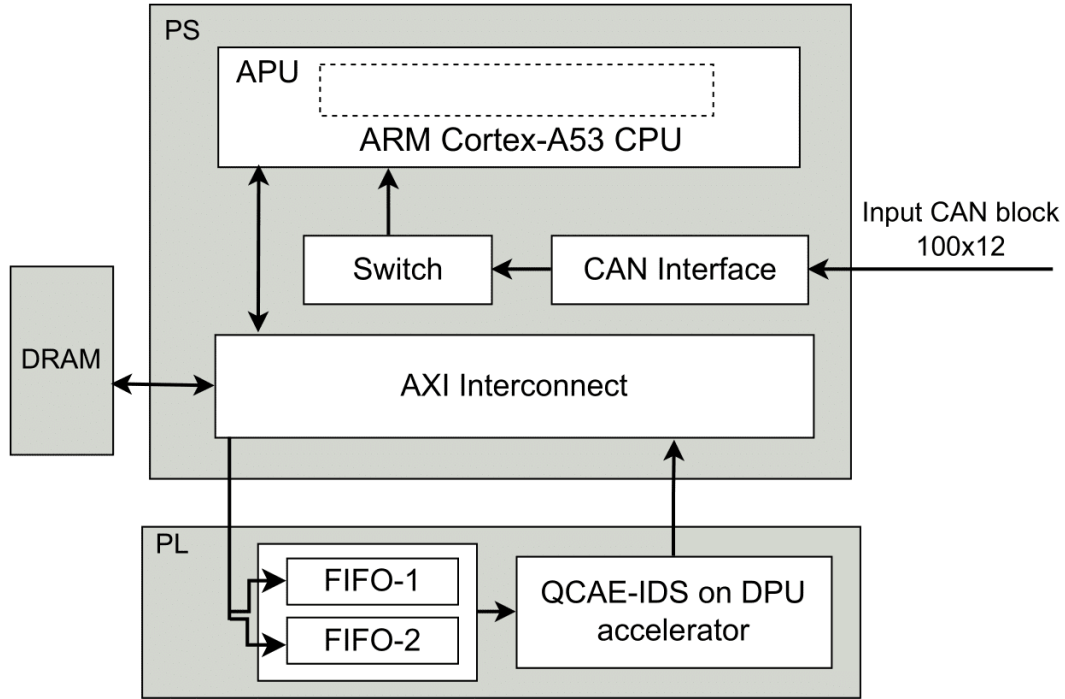


Fig. 4.5 Proposed system architecture of the integrated QCAE-IDS. The quantised CAE model is accelerated on the PL part of the FPGA device with the ping-pong buffers acting as the pre-processing block for the accelerator.

at 1.2 GHz peak. The DPUs use a 600 MHz DSP core clock and are configured to run concurrently for all the tests at startup. The Nvidia A6000 GPU has a base operating frequency of 1350 MHz.

For testing the IDS on FPGA with the ‘xmodel’ binary loaded, the performance is tested using the test split of all the attacks from both datasets. The inference accuracy is quantified by evaluating the precision, recall, and F1 rates of the QCAE-CH & QCAE-SA models. The per-block processing latency and average power consumption for performing IDS are also quantified on each incoming CAN message on the ECU. The results, in terms of inference accuracy and per-block processing latency, are compared against the state-of-the-art IDSs/IPs proposed in the research literature. The per-block processing latency and the per-inference energy consumption are compared when the INT8 version of the proposed model is executed on an NVIDIA Xavier GPU. In the case of schemes where inference is performed on a block of CAN messages, these metrics are used along with the block size for the comparison. The active power consumption of the models is compared against ML-IDS approaches in literature where power consumption has been reported.

Table 4.2 Inference accuracy metrics of CAE-CH pre- and post-quantisation (QCAE-CH) on all the attacks.

Attack	Model	Precision	Recall	F1	FPR	FNR
DoS	Pre-Q	99.56	99.89	99.73	0.37	0.11
	QCAE-CH	99.78	99.78	99.78	0.18	0.22
Fuzzy	Pre-Q	99.75	99.38	99.57	0.17	0.62
	QCAE-CH	99.63	99.38	99.50	0.25	0.62
RPM	Pre-Q	100	99.15	99.57	0	0.85
	QCAE-CH	100	99.07	99.53	0	0.93
Gear	Pre-Q	99.91	99.40	99.66	0.12	0.60
	QCAE-CH	99.91	99.40	99.66	0.12	0.60

Table 4.3 Inference accuracy metrics of CAE-SA pre- and post-quantisation (QCAE-SA) on all the attacks of the SA dataset.

Attack	Model	Precision	Recall	F1	FPR	FNR
Flooding	Pre-Q	100	99.28	99.64	0	0.7
	QCAE-SA	100	99.28	99.64	0	0.7
Fuzzy	Pre-Q	100	99.5	99.75	0	0.5
	QCAE-SA	100	99.5	99.75	0	0.5
Malfunction	Pre-Q	100	96.8	98.37	0	3.2
	QCAE-SA	100	94.24	97.04	0	5.7

4.5.1 Accuracy

To test the functional correctness of the model, the pre- & post-quantisation inference performance of the models is compared. An equal or higher performance is observed for the DoS & Gear spoofing attacks and a negligible drop in the F1 scores for the Fuzzing & RPM spoofing attacks pre- & post-quantisation for the QCAE-CH model as shown in table 4.2. For the SA dataset, an equal performance for the DoS & Fuzzing attacks is observed with a slight drop for the Malfunction attack. The QCAE-SA model performance is shown in table 4.3.

The performance of the models is first evaluated on the benign datasets to test for false positives. Out of 1000/150 blocks of messages which were tested (for QCAE-CH/QCAE-SA models), all of them were classified as benign with zero false positives. To demonstrate the ability to detect zero-day attacks, the classification performance of the models was tested on unseen DoS, Fuzzy, spoofing (Gear and RPM) and Malfunction attacks from the CH and SA datasets and compare them against the competing techniques (unsupervised learning techniques represented

Table 4.4 Confusion matrix capturing the classification results of the QCAE-CH on the attack-free (benign), DoS, Fuzzing, and Spoofing (Gear, RPM) attacks. (Note the model is trained only on benign messages and does not see any attack message during the training phase.)

Attack	Message Type	Predicted Normal	Predicted Attack
Benign	True Normal	1000	0
	True Attack	NA	NA
DoS	True Normal	1082	2
	True Attack	2	914
Fuzzy	True Normal	1190	3
	True Attack	5	802
Gear	True Normal	823	0
	True Attack	11	1166
RPM	True Normal	828	1
	True Attack	7	1164

Table 4.5 Confusion matrix capturing the classification results of the QCAE-SA on the attack-free (benign), Flooding, Fuzzing, and Malfunction attacks for the SA dataset. (Note the model is trained only on benign messages and does not see any attack message during the training phase.)

Attack	Message Type	Predicted Normal	Predicted Attack
Benign	True Normal	150	0
	True Attack	NA	NA
Flooding	True Normal	585	0
	True Attack	3	412
Fuzzy	True Normal	606	0
	True Attack	2	392
Malfunction	True Normal	594	0
	True Attack	24	393

in bold in the comparison tables) from the literature. Tables 4.4 and 4.5 capture the classification performance of the models in isolation across the test set as a confusion matrix. As discussed in section 4.4.2, the accuracies reported are obtained for the threshold value of 10 for the QCAE-CH model and a threshold of 73 for the QCAE-SA model. The misclassifications (false negatives) observed while testing on unseen attack datasets can be attributed to scenarios where the attack message flow closely resembles that of the normal messages within the observed block of messages at a given time.

The inference performance of the QCAE-CH model integrated within the ECU is compared against the state-of-the-art IDSs and IPSs proposed in the literature: GIDS [91], DCNN [18], MLIDS [31], HyDL-IDS [130] NovelADS [93], TCAN-IDS [92], IForest [95], MTH-IDS [23], GRU [94] and Rec-CNN [32] which are captured in tables 4.6 and 4.7, comparing them in terms of inference precision, recall, F1 score. In [23], the authors report an average F1-score of 96.3% for detecting zero-day attacks based on their stacked anomaly-based IDS. In comparison, QCAE-CH achieves an average F1 score of 99.6% for detecting the same set of unseen attacks. When compared to other unsupervised learning-based techniques, for the DoS attack QCAE-CH performs better than [91] by 1.7% in terms of the F1 score and only slightly less than [93] by 0.2%. In the case of **fuzzing** attack, the QCAE-CH model performs better than [91, 95] by 1.2% & 2% respectively in terms of the F1 score and only slightly less than [93] by 0.5%. In the case of **RPM spoofing** attack, the QCAE-CH model performs better than [91, 95] by 0.9% & 0.1% respectively in terms of the F1 score and only slightly less than [93] by 0.4%. In the case of **Gear spoofing** attack, the QCAE-CH model performs better than [91, 95] by 2.4% & 2.3% respectively in terms of the F1 score and only slightly less than [93] by 0.3%. It can be observed that the QCAE-CH model achieves comparable detection accuracy as state-of-the-art supervised learning models for all attacks as shown in tables 4.6 and 4.7 respectively.

Similarly, the inference performance of the QCAE-SA model is compared against the following IDSs proposed in the literature: XGBoost [136], Random Forest [138] and LSTM [137]. For the flooding attack, the QCAE-SA model performs better than [136] by $\approx 5\%$ in terms of F1 score, while being slightly less than [137, 138] by 0.4%. For the fuzzing attack, the QCAE-SA model performs better than [136] by 0.2% in terms of F1 score, while being slightly less than [137, 138] by 0.1% & 0.2% respectively. For the Malfunction attack, the QCAE-SA model suffers with a higher FNR which leads to it being less than [136–138] by 3% in terms of F1 score. The loss in performance for this attack can be attributed to the fact that these messages resemble the normal ECU communication closely. For the SA dataset, models could not be found that used unsupervised learning techniques. The detailed results of this model against these works can be found in table 4.8 respectively.

4.5.2 Inference latency

The processing latency of both models is quantified, starting from the arrival of the CAN message at the interface to determine the detection delay incurred by the

Table 4.6 Accuracy metric comparison (%) of the QCAE-CH accelerator against the reported literature on the DoS and Fuzzing attacks.

Attack	Model	Precision	Recall	F1	FPR
DoS	GIDS [91]	96.8	99.6	98.1	-
	DCNN [18]	100	99.89	99.95	-
	MLIDS [31]	99.9	100	99.9	-
	HyDL-IDS [130]	100	100	100	-
	NovelADS [93]	99.97	99.91	99.94	-
	TCAN-IDS [92]	100	99.97	99.98	-
	iForest [95]	-	-	-	-
	GRU [94]	99.93	99.91	99.92	-
	QCAE-IDS	99.78	99.78	99.78	0.18
Fuzzy	GIDS [91]	97.3	99.5	98.3	-
	DCNN [18]	99.95	99.65	99.80	-
	MLIDS [31]	99.9	99.9	99.9	-
	HyDL-IDS [130]	99.98	99.88	99.93	-
	NovelADS [93]	99.99	100	100	-
	TCAN-IDS [92]	99.96	99.89	99.22	-
	iForest [95]	95.07	99.93	97.44	-
	GRU [94]	99.32	99.13	99.22	-
	QCAE-IDS	99.63	99.38	99.50	0.25

approach. Table 4.9 compares the result against other approaches in the literature, which utilise different platforms (GPUs, Jetson edge accelerators, Raspberry Pi) and approaches (block of CAN messages v/s individual messages). The tightly integrated QCAE-CH model performs inference in 0.43 ms (B4096 DPU) for a block of 100 CAN frames, which is a $1.3\times$ improvement over the dedicated line-rate MTH-IDS ECU proposed in [23]. Similarly, the QCAE-SA model performs inference in 0.42 ms (B512 DPU) for a block of 100 CAN frames. The inference latency of the int8 CAE-CH & CAE-SA models is observed on the Jetson Xavier GPU to be 0.45 ms & 0.41 ms which makes it 4.4% slower than the proposed QCAE-CH model while being identical to the QCAE-SA model on the hybrid FPGA-based ECU. The ping-pong buffer at the PL allows the accumulation of 100 CAN IDs (approx 1 ms at full utilisation of CAN at 1 Mbps) in one buffer while the previous 100 IDs can be processed from the second buffer by the DPU, effectively hiding the processing latency within the reception delay of a block of messages. The proposed QCAEs hence achieve processing speeds that enable real-time detection of zero-day attack messages on high-speed critical CAN networks.

Table 4.7 Accuracy metric comparison (%) of the QCAE-CH accelerator against the reported literature on the RPM and Gear spoofing attacks.

Attack	Model	Precision	Recall	F1	FNR
RPM	GIDS [91]	98.3	99	98.6	-
	DCNN [18]	99.99	99.94	99.96	-
	MLIDS [31]	100	100	100	-
	HyDL-IDS [130]	100	100	100	-
	NovelADS [93]	99.9	99.9	99.9	-
	TCAN-IDS [92]	99.9	99.9	99.9	-
	iForest [95]	98.9	100	99.4	-
	QCAE-IDS	100	99.07	99.53	0
Gear	GIDS [91]	98.1	96.5	97.2	-
	DCNN [18]	99.99	99.89	99.94	-
	MLIDS [31]	100	100	100	-
	HyDL-IDS [130]	100	100	100	-
	NovelADS [93]	99.99	99.8	99.9	-
	TCAN-IDS [92]	99.9	99.8	99.9	-
	iForest [95]	94.7	100	97.3	-
	QCAE-IDS	99.91	99.40	99.66	0.12

Table 4.8 Accuracy metric comparison (%) of the QCAE-SA accelerator against the reported literature on the Flooding, Fuzzing and Malfunction attacks.

Attack	Model	Precision	Recall	F1	FNR
Flooding	XGBoost [136]	100	90	94.74	-
	Random Forest [138]	100	100	100	0
	LSTM [137]	-	100	100	0
	QCAE-SA	100	99.28	99.64	0.7
Fuzzing	XGBoost [136]	99.98	99.08	99.53	-
	Random Forest [138]	99.79	99.93	99.86	-
	LSTM [137]	-	99.95	99.96	.05
	QCAE-SA	100	99.5	99.75	0.5
Malfunction	XGBoost [136]	99.92	100	99.96	-
	Random Forest [138]	99.90	99.98	99.94	-
	LSTM [137]	-	100	100	100
	QCAE-SA	100	94.24	97.04	5.7

4.5.3 Power consumption and Resource Utilisation

The power consumption and resource overhead of the approach is measured to quantify the energy consumption per inference and the hardware resources incurred

Table 4.9 Inference latency comparison of QCAE-CH & QCAE-SA models against other state-of-the-art IDSs (supervised & unsupervised) reported in literature.

Models	Latency	Frames	Platform
GRU [94]	890 ms	5000 CAN frames	Jetson Xavier NX
MLIDS [31]	275 ms	per CAN frame	GTX Titan X
Rec-CNN [32]	117 ms	128 CAN frames	Jetson TX2
NovelADS [93]	128.7 ms	100 CAN frames	Jetson Nano
GIDS [91]	5.89 ms	64 CAN frames	GTX 1080
DCNN [18]	5 ms	29 CAN frames	Tesla K80
TCAN-IDS [92]	3.4 ms	64 CAN frames	Jetson AGX
MTH-IDS [23]	0.574 ms	per CAN frame	Raspberry Pi 3
QCAE-CH	0.43 ms	100 CAN frames	Zynq Ultrascale+
QCAE-SA	0.42 ms	100 CAN frames	Zynq Ultrascale+

on the device in table 4.10. The table shows the deployment of the models on three separate DPU configurations. The best tradeoff of latency, resource utilisation, and energy consumption for the QCAE-CH & QCAE-SA models is achieved with the B4096 & B512 DPU configurations respectively. For the QCAE-CH model, the B4096 DPU consumes $< 30\%$ LUT & $\approx 40\%$ DSP resources on the XCZU7EV device. Similarly, for the QCAE-SA model, the B512 DPU consumes $< 18\%$ LUT & $< 10\%$ DSP resources on the XCZU7EV device leaving enough resources for other custom accelerators on the PL part of the SoC. It is observed that the QCAE-CH & QCAE-SA model consume 4.89 W & 2.74 W when measured directly from the device's power rails (using the PYNQ-PMBus package) while performing inference and other tasks on the ECU (with Linux OS), which translates to 2.1 mJ & 1.1 mJ of energy consumed per inference by the models. Among other reported results in the literature, the QCAE-CH model results in a $\approx 2\times$ reduction in power consumption when compared to the GRU [94] model, which uses an Nvidia Jetson Xavier as a dedicated IDS node. It is also observed the per-inference energy consumption of the INT8 QCAE-CH & QCAE-SA models on the Jetson Xavier to be 2.1 mJ & 1.7 mJ when averaged over 10000 inference runs. The integrated ECU approach with QCAE-IDS achieves a $1.5\times$ reduction in energy consumed per inference for the QCAE-SA model while being identical to the QCAE-CH model. Hence, the models are either identical or better in terms of latency and energy utilisation over a dedicated Jetson Xavier-based IDS accelerator, making the proposed approach a more appropriate choice for deploying IDS in critical CAN systems.

Table 4.10 Performance scaling of the QCAE-CH & QCAE-SA model using different DPU configurations: impact on per message latency (ms), resource (%), & power consumption (W).

DPU	Latency (ms)		% Resource Utilisation					Power (W)		
	CH	SA	LUT	FF	DSP	BRAM	URAM	Idle	Active-CH	Active-SA
B512	1.1	0.42	17.34	11.10	7.18	16.51	12.5	2.61	3.11	2.74
B1152	0.77	0.33	19.53	13.92	13.08	19.55	33.33	3.12	3.56	3.33
B4096	0.43	0.30	27.17	25.03	40.74	35.42	47.92	4.36	4.89	4.59

4.6 Summary

In this chapter, two versions of quantised convolutional autoencoder-based IDSs are presented. The models are trained only on normal CAN messages capable of detecting zero-day attacks for automotive controller area networks in real time. The tradeoffs of using CAE's for this task have also been discussed. The proposed QCAE-CH & QCAE-SA models achieve state-of-the-art classification accuracy across multiple unseen attack vectors as well as a $1.3\times$ speed-up in processing latency and $\approx 2\times$ reduction in power consumption with 2.1 mJ & 1.1 mJ energy per inference when compared to the state-of-the-art IDSs. As seen with the QdCNN and QMLP case, the tight integration with the software function allows IDS capabilities to be consolidated at the ECU with complete isolation from the software function allowing IDS to be efficiently deployed in vehicles. The proposed IDS architecture can be expanded to a generalisable network anomaly detector architecture for detecting zero-day attacks at vehicular gateways that support high-speed networks like Automotive Ethernet.

A key constraint with the models so far was the requirement for sequences of CAN messages as the input feature to perform effective classification. However, this limits the ability of the IDS function to perform per-message analysis due to the inability of the ML models explored so far to inherently capture the time-series characteristics. The next chapter explores the use of models optimally suited for time-series prediction and develops design methodologies to address the multi-pronged optimisation challenge with automotive IDS.

4.7 Related Publications

- This chapter of the thesis was published as a paper, S. Khandelwal and S. Shreejith, "Real-Time Zero-Day Intrusion Detection System for Automotive

Controller Area Network on FPGAs," 2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP), Porto

- The QCAE-SA model developed in this chapter which was tested on the survival analysis dataset is part of a journal submission in ACM TECS currently under review.

Chapter 5

Quantised Recurrent Neural Network based IDS

5.1 Introduction & Contributions

The supervised and unsupervised models discussed in previous chapters operate on a window of CAN messages as its input feature. This sequenced data captures the existing dependency between adjacent CAN messages in the dataset using a memory/buffer structure in hardware. This workaround stems from the limitations of the models explored whereby these simple models are unable to capture such time-series dependencies between adjacent message frames natively. This window-based approach has two major issues:

- An external data-processing pipeline is required to capture this window of CAN messages, mirroring the window-based scheme used for training. The depth of this window is also experimentally determined, adding to the complexity of the design space exploration problem.
- A pessimistic labelling scheme is adopted for training and validation, whereby a malicious message in a window causes the entire window to be marked as malicious. This could result in numerous dropped frames in practical systems.

While recurrent models such as long short-term memory (LSTM) models can alleviate this problem, the higher computational complexity is a key barrier for constrained deployments. Unlike feed-forward models, quantised model training and mapping flows for recurrent models are not well-established and generalisable.

In this chapter, a generalisable mapping flow is developed to support custom quantised LSTM models by extending AMD's FINN tool flow. For deploying as the IDS, a quantised LSTM-based IDS architecture (referred to as QLSTM-IDS in this thesis) is developed, trained and deployed using the above-developed tool flow extensions. The LSTM layer acts as the input layer of the IDS capable of processing each message independently and capturing the time-series representation internally, in contrast to the block-based processing used in the previous two chapters. The tool flow extensions and custom transformations are used to generate a resource-efficient accelerator design which is synthesised to hardware using AMD's Vitis HLS tools with the Zynq Ultrascale+ ZCU104 platform as the target. To minimise the inference latency and software overheads, a bridged integration strategy is used to embed the IDS accelerator within the ECU. The proposed QLSTM-IDS achieves an intrusion detection accuracy exceeding 99.9% in multi-class classification across DoS/Flooding, Fuzzing, and Spoofing/Malfunction attacks across the two widely used attack datasets for automotive CAN. The implementation and integration optimisations enable a detection latency of 0.25 ms per message, allowing it to check and validate CAN messages in real-time on CAN bus, with a per-inference energy consumption of only 0.8 mJ, demonstrating its suitability for efficient and high-performance deployment in real-world vehicle environments.

The following are the major contributions of this chapter:

- A quantised LSTM-based IDS for CAN is presented, capable of per-message classification and detecting multiple attack vectors with the use of a single model.
- A generalisable implementation flow is developed by extending AMD's FINN tools to map the quantised LSTM model into a dataflow accelerator design.
- A custom integration approach is explored that reduces the latency overhead incurred by the coupled-accelerator scheme explored in previous chapters.
- The proposed model is evaluated on both the car hacking and survival analysis datasets to demonstrate its effectiveness with real-world intra-vehicular network data. The model achieves an average accuracy $> 99.9\%$ on both datasets across all attack vectors.
- The per-message processing latency and energy consumption of the model are compared against state-of-the-art IDSs from the literature. The results

show that the proposed model outperforms existing per-message classification IDSs, achieving a $2.29 \times$ improvement in per-message detection latency and a per-message energy consumption of 0.8 mJ.

- A discussion on the tradeoff balance achieved by the proposed QLSTM-IDS, making it particularly well-suited for deployment in real-world vehicular environments

Additionally, the training scripts, hardware build files, and deployment flow have been open-sourced in a publicly accessible GitHub repository: <https://github.com/RCSL-TCD/QLSTM-IDS>.

The rest of the chapter is organised as follows: Section 5.2 discusses the development flow of the QLSTM layer for deployment on FPGAs; Section 5.3 discusses the development of QLSTM layer based IDS model; Section 5.4 presents the performance of the model against state-of-the-art IDSs proposed in the literature and the chapter is concluded in Section 5.5.

5.2 QLSTM layer development

This section begins with the introduction to RNNs followed by the development steps of the front end for the QLSTM layer, FINN compiler transformations and concludes with its corresponding hardware implementation.

5.2.1 Recurrent Neural Networks

RNNs were introduced in the 1990s primarily for sequence prediction tasks, such as speech recognition and natural language processing. Their key advantage is the ability to maintain hidden states that capture information from previous inputs [143]. This ability to model temporal dependencies makes them especially useful in applications where context and patterns evolve over time. Despite their advantages, RNNs were challenging to train due to issues such as the vanishing gradient problem, which were later addressed with advanced architectures like LSTMs and GRUs. LSTMs and GRUs have since found widespread applications in many sequence prediction tasks. The compute equations (gates and states) for state computations in LSTMs are shown below:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (5.1)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (5.2)$$

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (5.3)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (5.4)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (5.5)$$

$$h_t = o_t \odot \tanh(C_t) \quad (5.6)$$

where σ represents the sigmoid activation function, $\tanh$ is the hyperbolic tangent activation, and $\odot$ denotes element-wise multiplication. Due to the computationally intensive nature of these operations, LSTMs are often accelerated using bespoke hardware implementations on FPGAs to support real-time applications. In the next subsection, discusses the development of the ONNX layer development for the QLSTM layer.

5.2.2 Software front end

The Open Neural Network Exchange (ONNX) is an open standard designed to represent machine learning models. It defines a comprehensive set of operators (like `Matmul`, `ReLU`), which serve as the fundamental building blocks for constructing machine and deep learning models. ONNX aims to provide a unified format that fosters interoperability, enabling seamless integration and interaction between various AI tools and frameworks [144]. The ONNX representation also serves as a foundation for various AI hardware development frameworks, such as FINN, providing input for their compilers to apply transformations and map models to pre-built hardware layers. Additionally, ONNX supports modelling quantisation operators, such as `QuantizeLinear` and `DequantizeLinear`, enabling the representation of quantised neural networks (QNNs) with standard ONNX operators.

ONNX, however, traditionally supports operators that are forward-compute in nature and does not support many operators for recurrent computations. With the increasing demand for recurrent neural networks, ONNX introduced the `scan` operator to support their modelling [145]. This operator processes one or more input tensors iteratively to generate zero or more output tensors. It is inspired by general recurrences and functional programming constructs, such as `scan`, `fold`, `map`, and `zip`. During each iteration, the inputs to the operator consist of the current values of its state variables and the specific element of the input tensor being processed. Its outputs comprise the updated state variables and one or more output tensors. The computation performed in each iteration is defined by the operator's `body`

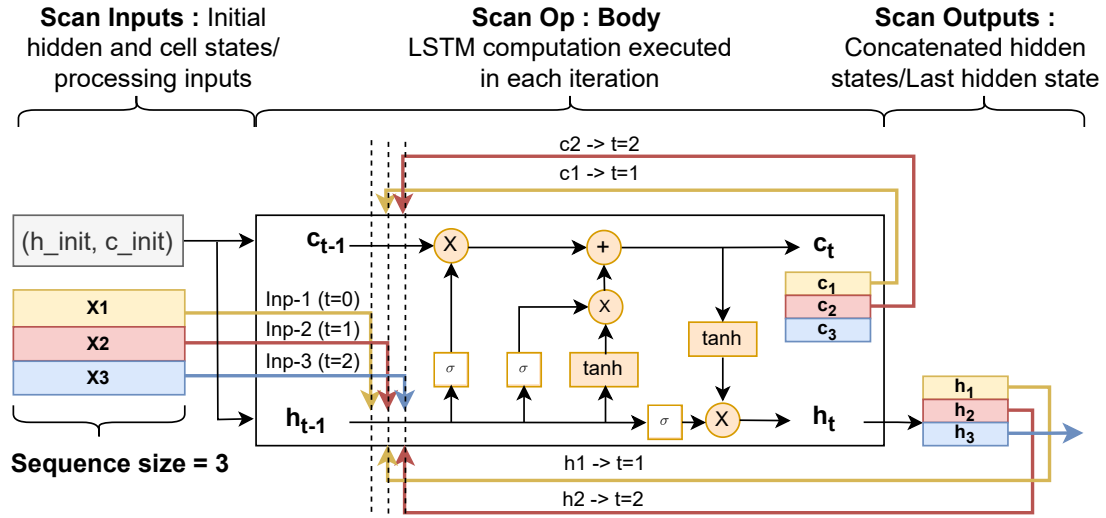


Fig. 5.1 The figure illustrates how the **Scan** operator replicates the recurrent hidden state computations in an LSTM layer. The yellow, red, and blue arrows highlight the pathways for input data processing and state updates at each time step for every input in the sequence.

attribute that encodes a computational graph. The operator has unique properties that differentiate it from other ONNX compute nodes:

- It allows the state variable to be updated after each iteration, enabling the updated state to be used in the processing of the next input.
- It processes inputs sequentially, either row by row or column by column, continuously updating the hidden state for each input while storing all intermediate outputs as hidden states.

This container operator is leveraged to encapsulate the LSTM compute graph within its **body**, effectively modelling the recurrent computational structure of the LSTM. The implementation is functionally verified by comparing it with the **QuantLSTM** layer implementation by **brevitas**. Figure 5.1 illustrates the internal workings of the scan operator in the context of the LSTM compute graph for three time steps.

5.2.3 FINN-Compiler Optimisations

Once the LSTM layer operation is functionally verified using the **scan** operator, the next step is to build its ONNX implementation using **QuantizeLinear** and **DequantizeLinear** operators and convert this **QuantizeLinear-Clip-DequantizeLinear** (QCDQ) computation graph into a set of pre-built hardware building blocks available in the **finn-hlslib** repository. This process, referred to as **streamlining** in

FINN, involves mapping the ONNX graph (QCDQ definition) to a functionally equivalent computation graph composed of operators that can be implemented in hardware. The first step in streamlining is converting the ONNX QCDQ graph into its QONNX representation. QONNX extends the ONNX framework by introducing custom quantisation operators (**Quant**, **BipolarQuant**, **Trunc**) to support arbitrary-precision uniform quantisation [146]. This conversion is achieved by applying the **ConvertQCDQtoQONNX** transformation to the graph. This transformation primarily converts the QCDQ operators into **Quant** nodes, as shown below:

$$\text{QuantizeLinear} + \text{Clip} + \text{DequantizeLinear} \rightarrow \text{Quant} \quad (5.7)$$

After obtaining the QONNX representation of the LSTM computation graph, the next step is to convert it into the FINN-ONNX representation. This conversion is performed using the **ConvertQONNXtoFINNONNX** transformation, which replaces QONNX operators and activations with the **matmul**, **multithreshold** operators, as shown below:

$$\text{Tanh} + \text{Quant} \rightarrow \text{Multithreshold} + \text{Mul} + \text{Add} \quad (5.8)$$

The LSTM computation graph also includes **sigmoid** and **tanh** activations, which are not directly supported in the above transformation. However, since they are both monotonically increasing, they can be modelled using **multithreshold** operators as shown in figure 5.2. The **sigmoid** and **tanh** activations are thus translated to **multithreshold** operations in the graph to attain compatibility with downstream FINN transformations. After the FINN-ONNX conversion is achieved and functionally verified, the resulting compute graph exclusively comprises operators compatible with existing FINN transformations.

The primary goal at this stage is to minimise floating-point operations in the computation graph and eliminate operators by fusing/absorbing them wherever possible. Despite applying all available transformations in the FINN compiler, certain unique computation patterns in the graph could not be streamlined. To address these patterns, the existing transformations are either modified or new ones are created and integrated into the FINN compiler. Along with these new transformations, a few more were customised to achieve the removal of all floating-point operations from the compute graph like **AbsorbSignBiasIntoMultithreshold**. 5.3 illustrates examples where our transformations allow the floating point operations to be absorbed into the

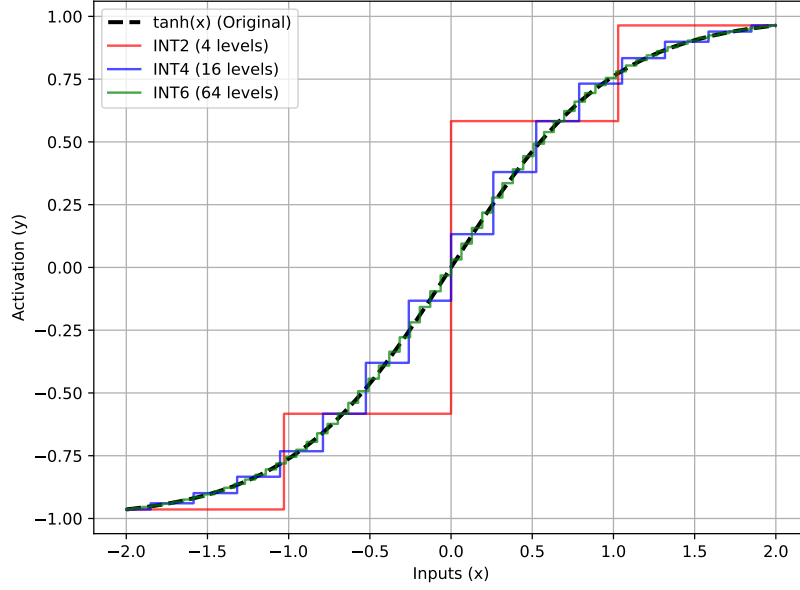
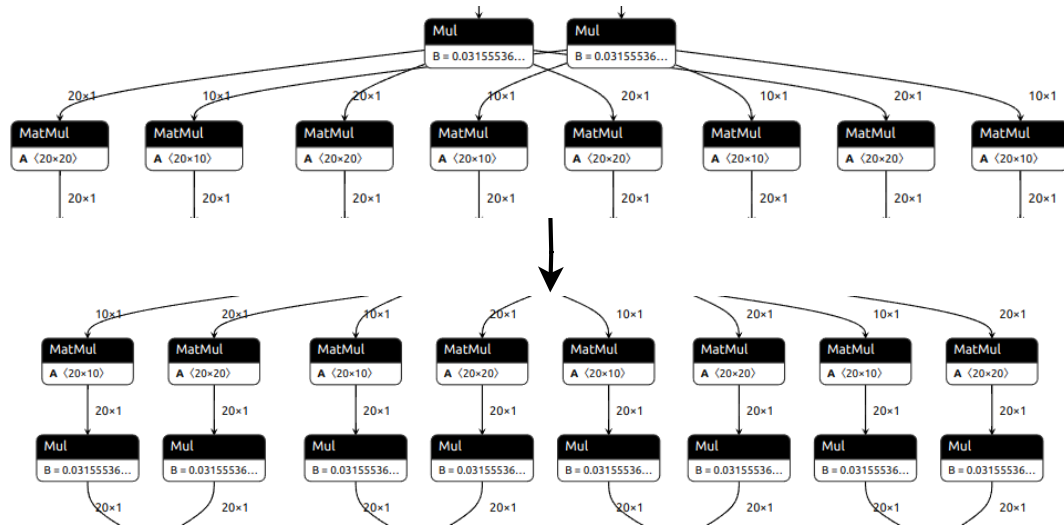


Fig. 5.2 The figure illustrates how the Multithresholding operator approximates the tanh activation function over a defined input range for INT2, INT4 and INT6 bitwidths.

nearest multi-thresholding operator. These transformations allow the LSTM graph to be fully streamlined, where all computations can be represented by (hardware) library functions in FINN and can hence be composed into hardware using the FINN-HLS backend. With the compute graph for the LSTM layer streamlined and all floating-point operations eliminated, the final step is then to build its hardware layer.

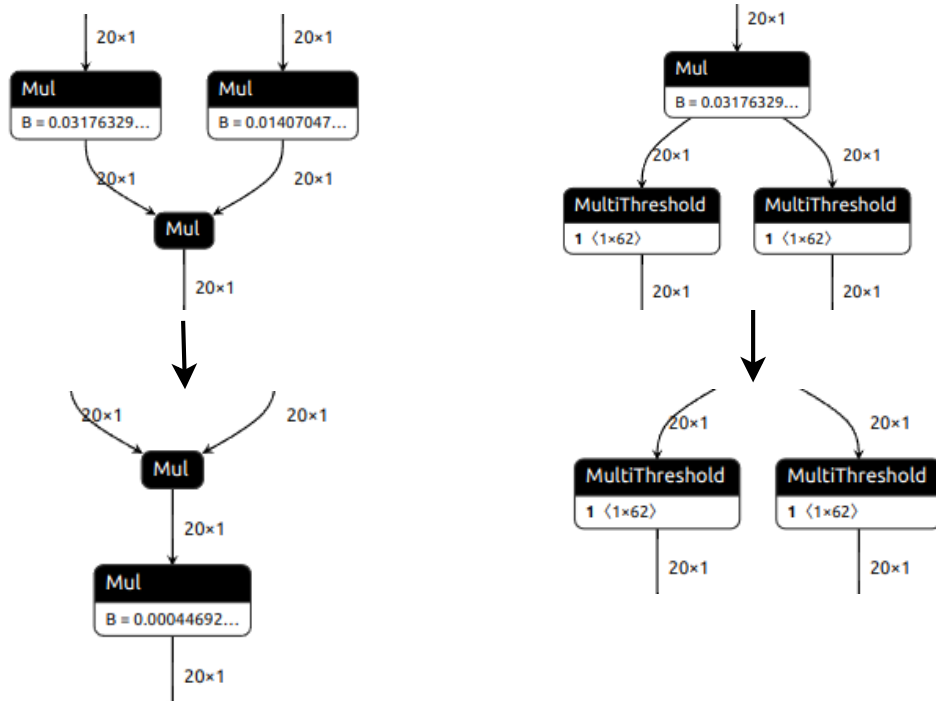
5.2.4 QLSTM HLS Backend

The backend flow maps the QCDQ representation to a functionally equivalent graph consisting of hardware blocks that implement the operators/transformations. This produces a quantised LSTM (Q-LSTM) computation graph with the following operators: **Threshold**, **MatMul**, **Elementwise Add**, and **Elementwise Mul**. As before, pre-existing hardware blocks were leveraged available in `finn-hlslib` which were developed for mapping feed-forward networks (using Vitis HLS) to map the forward computation path of LSTM layers. In the case of recurrent (loopback) connections, a standard `for` construct is utilised to map these to hardware blocks, whose bounds are determined by the input sequence length of the model. This loop structure introduces sequential dependencies in the LSTM computation graph and imposes a limitation on parallelisation (through unrolling). A longer input sequence length



MoveScalarMulPastMatMul() :

Added support for moving scalar Mul past each consumer Matmul in the fork.



MoveLinearPastEltwiseMul() :

Added support for moving join Mul's past element wise mul operator

AbsorbMulIntoMultithreshold() :

Added support to absorb scalar mul into multiple multithreshold nodes

Fig. 5.3 The figure illustrates how the proposed transformations allow the floating point operations to be absorbed into the nearest multi-thresholding operator.

can increase the initiation interval (II) of the loop, which could be a constraint for real-time applications. Utilising the building blocks enabled me to design a generic hardware layer with customisable parameters such as data types, input size, hidden layer dimensions, and sequence length. Once the design is complete, the accelerator is synthesised and exported as an independent IP block. Figure 5.4 illustrates the end-to-end development and deployment flow of an LSTM-based model created with FINN, starting from training the quantised variant using *brevitas* to its final deployment on hardware.

5.3 QLSTM-IDS

In this section, the QLSTM-based IDS model is developed, the exploration process is outlined, and the experimental results are presented.

5.3.1 Model Development

To arrive at a model architecture to perform multi-class classification on a per-message basis, several parameter combinations were evaluated. Through extensive experimentation with **LSTM** (feature generation) + **Dense** (classification) model configurations, it was observed that chaining an LSTM layer's output with a shallow dense model proved to be an ideal configuration to achieve highly accurate multi-class classification across the attack vectors. A high-level overview of the model's architecture is shown in Figure 5.5. The entire CAN packet (10 bytes including headers) is mapped into 20 **uint4** type values, which form the input to the model. The input layer of the model is an LSTM layer configured to process sequences of 10 inputs at each clock edge. The LSTM layer contains 20 hidden units and computes hidden states that capture the time-series relation within CAN frames to detect malicious messages. The layer uses a **lookback** parameter of 1, confining the lookback to the previous half-segment read into the LSTM model from the current message. The output from the LSTM layer is fed to a dense model with three fully connected layers, with 64, 32, and (4/5) neurons respectively. The number of neurons in the last layer depends on the number of attack classes in the testing dataset. ReLU activation functions are applied between the Dense layers to introduce non-linearity, with the final output layer producing the multi-class classification result. The final model comprises of $\approx 6,200$ training parameters. Standard **PyTorch** functions and

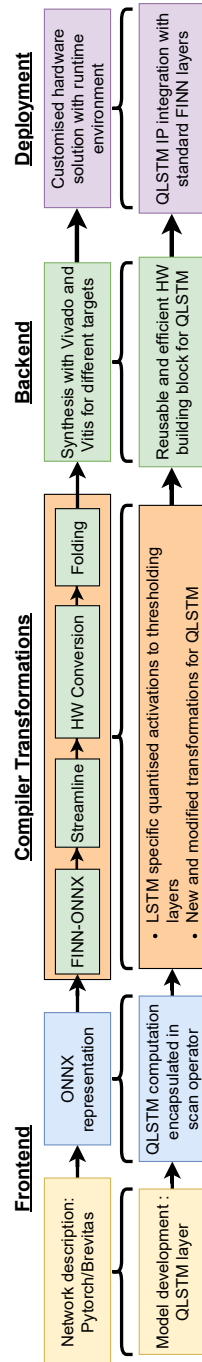


Fig. 5.4 This figure illustrates the different stages in the overall deployment workflow for generalised quantised LSTM layer (with mixed quantisation support) based models as an extension of the FINN framework for FPGAs. This figure is inspired by Figure 6 in [147].

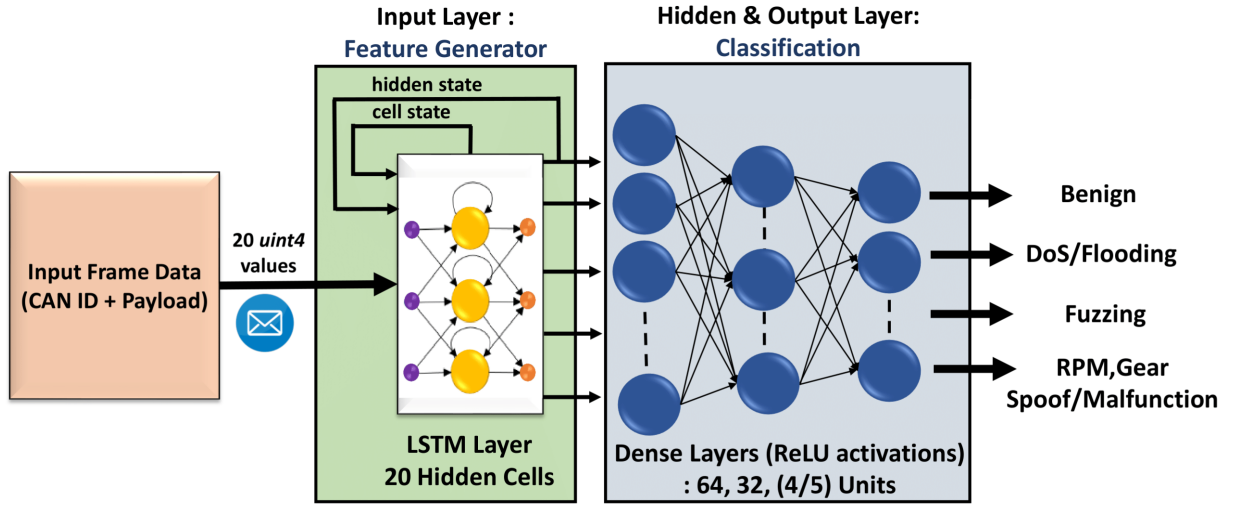


Fig. 5.5 The figure shows the network architecture of the proposed LSTM-IDS. The CAN input is packed into 20 uint4 values as the input to the LSTM layer. The classified outputs are packed into a byte-wide data.

training framework were used for experimenting with different configurations and for training/testing the initial models.

5.3.2 Dataset, Training & Deployment

5.3.2.1 Dataset Descriptions:

As before, we use the open Car Hacking dataset and the Survival Analysis datasets for training and validating the model. Both datasets were pre-processed to filter out features (CAN Packet Data) used for the training and testing flows. All the attack datasets were also preprocessed to convert the CAN ID + Payload features used as inputs to the model to the **uint4** datatype as this datatype allowed us to retain the highest accuracy through the model compression stages. For the Car Hacking Dataset, the training set consisted of 10 million CAN frames (84.3%), the validation set contained 1.24 million CAN frames (10.4%), and the test set included 620,200 CAN frames (5.2%). These subsets were constructed by selecting an equal number of messages from the dataset's four attack categories: DoS, Fuzzing, RPM-Spoof, and Gear-Spoof. In the case of the Survival Analysis Dataset, all datasets captured from the same car model were combined to train the model for multi-class classification. However, it was observed that the number of CAN frames captured from the three vehicles (as provided by the dataset) varied significantly. To ensure balanced datasets

between benign and attack messages, the following train, validation, and test splits were applied for each model:

- Sonata model: 260,000 CAN frames (67.5%) for training, 50,000 CAN frames (12.9%) for validation, and 75,000 CAN frames (19.4%) for testing.
- Soul model: 360,000 CAN frames (70.5%) for training, 60,000 CAN frames (11.7%) for validation, and 90,000 CAN frames (17.6%) for testing.
- Spark model: 172,000 CAN frames (67.7%) for training, 32,000 CAN frames (12.5%) for validation, and 50,000 CAN frames (19.6%) for testing.

Due to the relatively smaller size of the Survival Analysis Dataset, a larger proportion of the data was allocated to testing to effectively demonstrate the efficacy of the proposed model’s training and performance.

5.3.2.2 Training procedure

After arriving at the model architecture (LSTM + Dense) by observing its efficacy at floating point precision (fp32), mixed-precision quantisation (weights/activations) was explored to reduce the model’s compute and memory requirements. The LSTM and Dense layers were quantised from fp32 weights/activations to a quantised version with 8-bit quantised weights and 6-bit quantised activations (including the internal activations of the LSTM layer). A DSE on the model’s weight bit-widths was not conducted, as the corresponding matrix multiplication operations—into which these weights are synthesised in hardware—do not significantly impact resource utilisation. However, a DSE for activation bit-widths was essential, as resource usage increases exponentially with activation precision. This is because activations are implemented as MultiThreshold blocks in hardware, which scale non-linearly with bit-width. Figure 5.6 presents the average training and validation loss for activation bit-widths ranging from 2 to 8. The 2-bit activation model was excluded from the plot, as it exhibited rapid overfitting during training. Ultimately, 6-bit activations were selected with a combination of 8-bit weights, offering the lowest validation loss across all configurations trained for 50 epochs. The quantised version of this model is trained with AMD’s QAT library, **brevitas**.

For all the datasets, the **Adam** optimiser was used with the **CrossEntropyLoss** as the loss function for multi-class classification. The **CrossEntropyLoss** loss function was chosen as they are known to improve the training flow when dealing with a multi-class classification problem with an unbalanced training set like in this case.

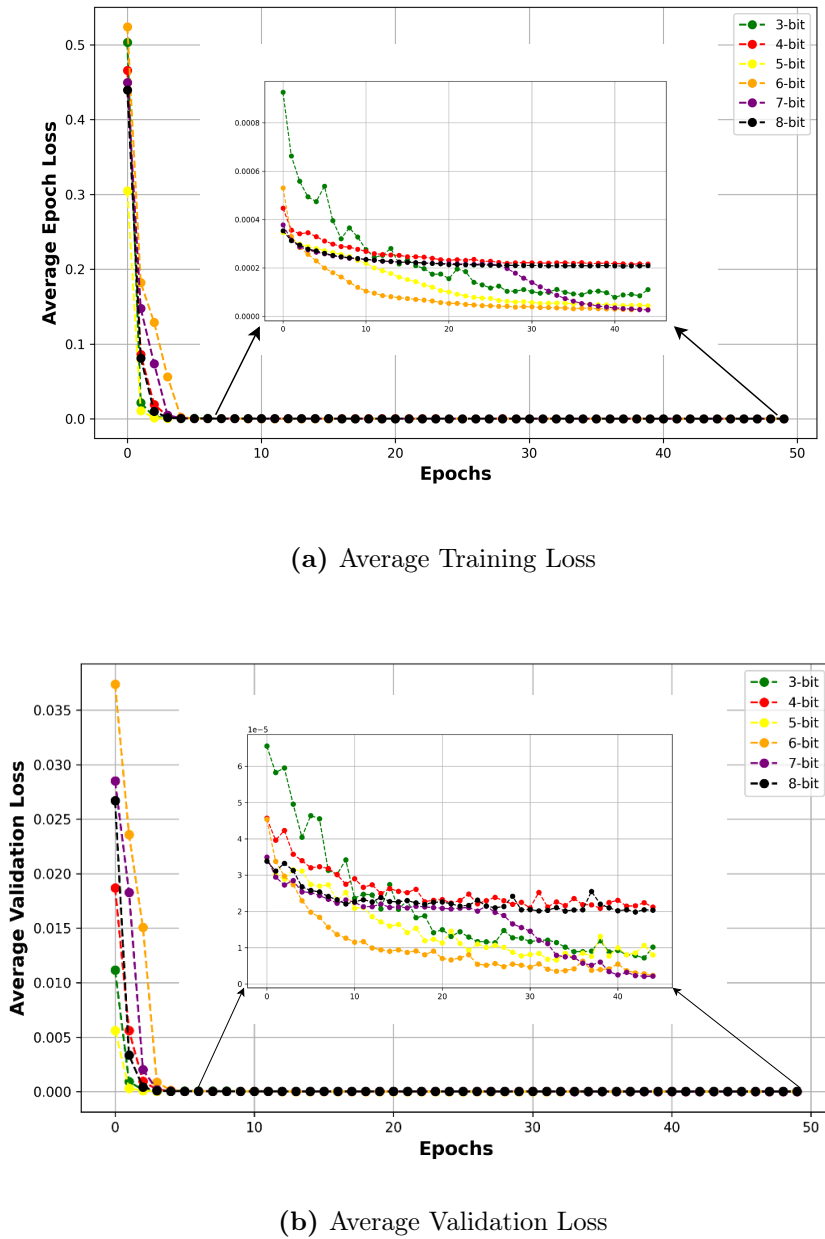


Fig. 5.6 The figure illustrates the average training and validation loss curves for the QLSTM-IDS during its training for different activation precisions using the car hacking dataset. To better highlight the differences, the figure provides a magnified view of epochs 5 to 50.

The `learning_rate` was set for all the training flows to 0.0001. A slower `lr` helps to improve model convergence as has been shown by many studies [125]. With the car hacking dataset, the model converged quicker at 50 epochs with a batch size of 2000.

In the case of the survival analysis dataset, the convergence was slower due to the smaller training set, requiring the model to be trained for 100 epochs with a batch size of 200 to achieve the best accuracies. Dropout layers were incorporated between computational layers to mitigate overfitting, enhancing the model's generalisation and training efficiency. Additionally, weight clipping techniques were employed during the training to address gradient explosion in the LSTM layer, ensuring stable gradient updates and improved convergence. Figure 5.7 presents the training and validation loss curves for the QLSTM-IDS model trained on the four attack datasets. The training process exhibits stable convergence, with the best-performing models for each dataset (Car Hacking, Sonata (SA), Soul (SA), and Spark (SA)) selected at epochs 29, 64, 75, and 83, respectively, based on the lowest validation loss.

5.3.2.3 Generating the dataflow accelerator

Generating the dataflow hardware model for generalised LSTM models is not natively supported by open-source tools such as FINN. However, it has been shown that custom implementations of uniformly quantised LSTMs can be achieved using hardware building blocks available in FINN's high-level synthesis library (`finn-hlslib`) as shown in [148]. Building on this approach, the trained mixed precision QLSTM-Dense model was converted to the ONNX intermediate representation (IR). Brevitas export flow does not support quantised LSTM layers and a custom handler function was built to generate the ONNX model for the QLSTM model. This export representation captures the computations, weights and scales/zero-points of the quantisers that were used to achieve the quantisation, which was required by the mapping flow to generate the hardware dataflow block. To map the ONNX nodes to the available hardware building blocks, certain transformations provided by FINN were reverse-engineered and new transformations were created in the FINN compiler to map the computations in the ONNX graph to `finn-hlslib` hardware blocks. In this step, the graph nodes that do not have a one-to-one representation in the library were transformed using the custom transformations to represent them using FINN-compatible hardware blocks. Subsequently, the updated computation graph was passed to Vitis HLS (v2022.2) flow and linked with the FINN libraries to map these blocks using the hardware blocks available in the FINN backend. Pragma's were used to unfold loops for parallel processing as well as to map the input and output interfaces of the model as AXI stream allowing for seamless integration with other IP blocks and/or

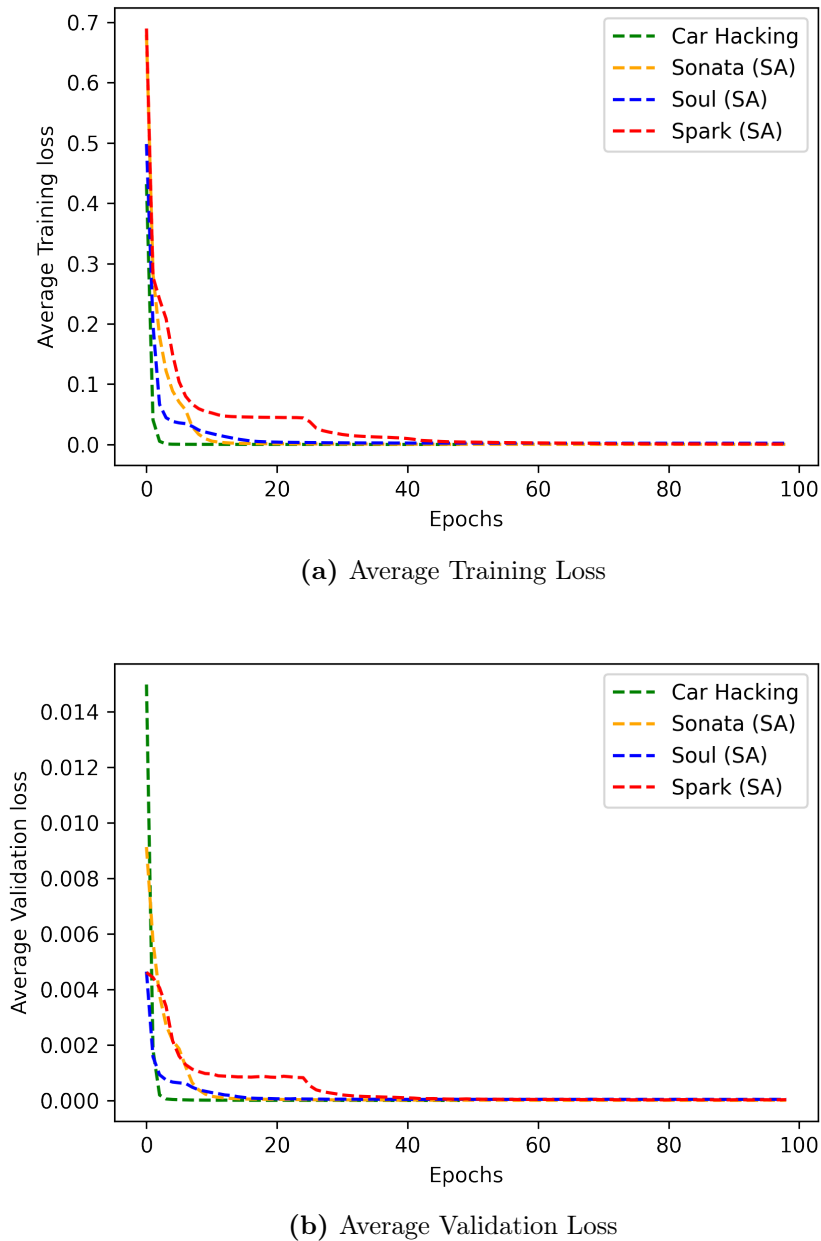


Fig. 5.7 The figure illustrates the average training and validation loss curves for the QLSTM-IDS during its training on the four attack datasets from the car hacking and survival analysis datasets.

processing systems on a Zynq device (the hardware description script can be accessed via the [GitHub repository](#)).

For testing, the IP was integrated into a PYNQ overlay design with a data pre-processor which converts byte-wide CAN message data into the `uint4` representation

used by the model. The PYNQ overlay includes interface wrappers to move data from/to the IP from software on the ARM cores, to aid with the evaluation. The Zynq Ultrascale+ XCZU7EV device on the ZCU104 development platform was chosen as a target platform for the implementation. For testing the I/O interface and the final model, the PYNQ interface was used to write data to the model via the pre-processor FIFOs and to read the classified result from the accelerator IP. This flow was chosen as it allows us to determine the classification accuracy, detection latency, and energy consumption of the hardware implementation of the model while embedding the IDS function in an architecture that mimics a standard ECU. While the PYNQ flow introduces software overheads, it provides a seamless interface for moving data between the ARM cores and logic and provides API functions to probe on-board power rails directly from software at runtime.

5.3.3 Bridged IDS Integration

Figure 5.8 illustrates the proposed architecture that integrates the IDS function closer to the network interface to reduce software overheads and latency while retaining software-based control. The Zynq Ultrascale+ XCZU7EV device on the ZCU104 platform is chosen as the target device.

In a deviation from previous integration models, the IDS was directly integrated by bridging the CAN controller's interface to the PS. This can be done either by tapping into the AXI dataflow or by replicating the interface to pass the data to the IDS. The second option is more appealing since the open-source CAN controller IP is used in the model implementation would allow the interface and internal FIFOs can be replicated to improve the throughput, at the cost of slightly higher resource cost. The first case is implemented for this test, where the AXI bus lines between the controller and the AXI interconnect are tapped to feed the data to the pre-processing FIFO, as this could be replicated in the case of custom (proprietary) CAN controllers. The pre-processing FIFO receives the byte-wide message data from the CAN controller and converts it to the **uint4** type required by the model using an asymmetric width interface. Streaming interfaces are used for the FIFO and the IDS to minimise overheads. The FIFO also aids in clock domain crossing, from the slower clock domain used by the CAN controller to the faster clock used by the QLSTM implementation. To support the hardware testing, a datapath was added that allows the software task to load the message to the IDS controller through the AXI interconnect. A multiplexer is used to switch the input path from the CAN

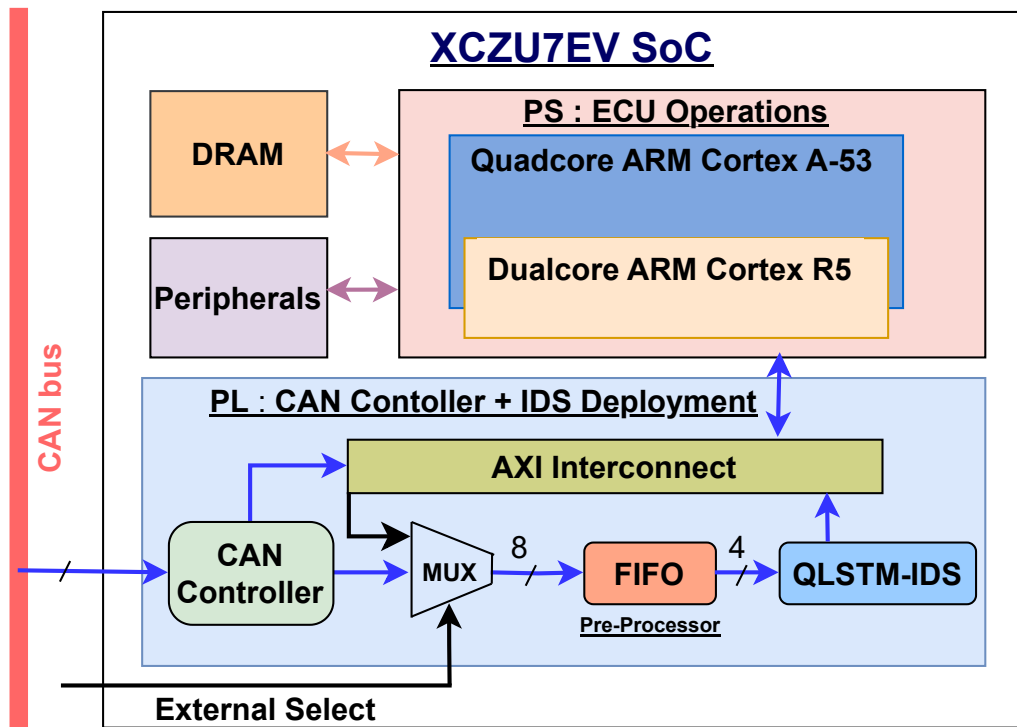


Fig. 5.8 The figure illustrates the ECU architecture modelled on a Zynq Ultrascale+ hybrid SoC and the path of a CAN message upon reception in this architecture.

In this architecture, the processing system (PS) functions as the ECU, executing automotive applications. The blue arrows indicate the path taken by the CAN message from the CAN bus to the software tasks on the PS. The IDS processing is integrated within the programmable logic (PL) section of the XCZU7EV device. The IDS can receive the CAN message directly from the controller (at the same instance as it is read by the PS) or forwarded by the PS. A multiplexer is used to control this at run-time, with the select line connected to an external peripheral on the board.

controller to the AXI interconnect and is controlled by an external signal (mapped to a switch input on the ZCU104 board). Once the IDS decision is available, the result is communicated to the processing system through the AXI interconnect. Based on the IDS's decision, the ECU within the PS can take appropriate preventative actions in response to detected attacks like dropping the message or triggering safety protocols, ensuring robust system security.

5.4 Experimental Results

To demonstrate the classification performance of the QLSTM-IDS model it is tested with both the open-car hacking & survival analysis dataset by replaying them on the ZCU104 platform. The model performance is compared in terms of accuracy metrics precision, recall and F1 score against the following state-of-the-art IDSs proposed in the research literature: BIDS [149], QMLP-IDS [150], DCNN [18], TCAN-IDS [92], GRU [94], HybridSecNet [151], LSTM [137], Transformer [132], NLP [152], Random Forest [138]. The detection latency, resource utilisation & actual energy consumption of the QLSTM model on the FPGA are quantified. The observed results are compared with other works in the literature where these results have been reported. The QLSTM-IDS accelerator was synthesised with an operating frequency of 150 MHz. The PYNQ image is used on the ARM cores with PetaLinux tools providing a seamless interface for data movement. The heavy OS layers allow us to compute the worst-case latency in the presence of software and driver overheads. The APIs in the PYNQ libraries allow is used to directly monitor and isolate the power rails of the fabric from the ARM cores, while the model runs inference.

5.4.1 Accuracy

Car Hacking dataset: With the Car Hacking dataset, the QLSTM-IDS model achieves an impressive overall classification accuracy of 99.998%. The test set includes 620,000 messages, consisting of 502,591 benign, 25,760 DoS, 32,361 Fuzzy, 32,385 RPM-Spoof, and 27,103 Gear-Spoof messages. The classified outputs are captured using a multi-class confusion matrix shown in Figure 5.9, which reveals that the model only misclassified 11 messages across the entire test set. From the results, key performance metrics such as precision, recall, F1 score, and FNR are calculated. These results are compared with several state-of-the-art IDS models that have used the same dataset for training and testing, including DCNN [18], NovelADS [93], TCAN-IDS [92], GRU [94], HybridSecNet [151], BIDS [149], and CQMLP-IDS [150]. The comparative results from the referenced studies are presented in Table 5.1 & Table 5.2 respectively. For the DoS attack, the proposed QLSTM-IDS model detects all attack messages, achieving an F1 score superior to DCNN [18], NovelADS [93], GRU [94], BIDS [149], and CQMLP-IDS [150] by margins of 0.05%, 0.06%, 0.08%, 0.37%, and 0.1%, respectively, while performing on par with TCAN-IDS [92]. In the case of Fuzzy attack, the QLSTM-IDS model outperforms DCNN [18], TCAN-IDS [92], GRU [94], BIDS [149], and CQMLP-IDS [150] in F1 score by 0.2%, 0.8%,

Table 5.1 Accuracy metric comparison (%) of the QLSTM-IDS accelerator against the reported literature on the DoS and Fuzzing attacks for the car hacking dataset.

Attack	Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
DoS	DCNN [18]	100	99.89	99.95	0.13
	NovelADS [93]	99.97	99.91	99.94	-
	TCAN-IDS [92]	100	99.97	99.98	-
	GRU [94]	99.93	99.91	99.92	-
	BIDS [149]	99.50	97.73	98.63	-
	QMLP-IDS [150]	99.92	99.88	99.90	0.11
	QLSTM-IDS	100	100	100	0
Fuzzy	DCNN [18]	99.95	99.65	99.80	0.5
	NovelADS [93]	99.99	100	100	-
	TCAN-IDS [92]	99.96	99.89	99.22	-
	GRU [94]	99.32	99.13	99.22	-
	BIDS [149]	100	99.38	99.69	-
	QMLP-IDS [150]	99.93	99.69	99.81	0.27
	QLSTM-IDS	99.99	99.98	99.98	0.02

0.8%, 0.3%, and 0.2%, respectively, and performs equivalently to NovelADS [93]. For the **RPM-Spoofing** attack, the model again successfully detects all attack messages and achieves an F1 score improvement over NovelADS [93], TCAN-IDS [92], GRU [94], and BIDS [149] by 0.1%, 0.1%, 0.4%, and 0.14%, respectively, while performing comparably to DCNN [18] and CQMLP-IDS [150]. Similarly, for the **Gear-Spoofing** attack, the QLSTM-IDS model identifies all attack messages and achieves a better F1 score than DCNN [18], NovelADS [93], GRU [94], and BIDS [149] by 0.06%, 0.1%, 0.4%, and 0.1%, respectively, performing on par with TCAN-IDS [92]. Considering a multi-class scenario, HybridSecNet [151] reports average precision, recall, and F1 scores of 99.57%, 99.58%, and 99.57%, respectively, across all attack types. In comparison, the QLSTM-IDS model achieves average scores of 99.99% across all three metrics, surpassing HybridSecNet by 0.4% in each metric. These metrics demonstrate that QLSTM-IDS provides highly efficient multi-attack detection on the Car Hacking dataset, outperforming several state-of-the-art approaches.

Survival Analysis Dataset : The survival analysis dataset, as discussed in the previous section, includes attack datasets for three different car models: HYUNDAI Sonata (2010), KIA Soul SUV (2015), and CHEVROLET mini/compact vehicle Spark (2015). The model is evaluated against the attack datasets from each of these models.

Table 5.2 Accuracy metric comparison (%) of the QLSTM-IDS accelerator against the reported literature on the RPM and Gear attacks for the car hacking dataset.

Attack	Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
RPM	DCNN [18]	99.99	99.94	99.96	0.11
	NovelADS [93]	99.91	99.9	99.91	
	TCAN-IDS [92]	99.9	99.9	99.9	
	GRU [94]	99.95	99.31	99.63	
	BIDS [149]	99.99	99.73	99.86	-
	QMLP-IDS [150]	99.96	100	99.98	0
	QLSTM-IDS	100	100	100	0
Gear	DCNN [18]	99.99	99.89	99.94	0.12
	NovelADS [93]	99.89	99.93	99.91	-
	TCAN-IDS [92]	99.99	99.98	99.98	-
	GRU [94]	99.95	99.31	99.63	-
	BIDS [149]	99.99	99.84	99.91	-
	QMLP-IDS [150]	-	-	-	-
	QLSTM-IDS	100	100	100	0

The test set for each model consists of the following number of messages: Sonata - 75,000 (Normal: 59,493, Flooding: 6,735, Fuzzing: 4,780, Malfunction: 3,992); Soul - 90,000 (Normal: 73,511, Flooding: 8,705, Fuzzing: 6,327, Malfunction: 1,457); and Spark - 50,000 (Normal: 40,528, Flooding: 7,011, Fuzzing: 1,480, Malfunction: 981). The multi-class confusion matrices generated from the classification output of the model across different vehicle models in the survival analysis dataset are shown in Figure 5.10, 5.11 & 5.12 respectively.

From figures 5.10–5.12, it can be observed that the model output produces 7, 50, and 149 misclassifications across the test sets for Sonata, Soul, and Spark cars, respectively. As before, the precision, recall, F1 scores, and FNR for each attack set are computed and the findings are compared with state-of-the-art models that have utilised this dataset, including XGBoost [136], LSTM [137], Transformer [132], NLP [152], and Random Forest [138]. Tables 5.3, 5.4, and 5.5 show the comparative performance of the QLSTM-IDS against the above state-of-the-art models across the three vehicle datasets, Sonata, Soul and Spark respectively.

On the **Sonata** subset, the QLSTM-IDS demonstrates its excellent ability to generalise across different attack vectors by achieving very high detection accuracy and zero misclassifications across all attacks. Compared to the state-of-the-art schemes, the model achieves a 5.3% higher F1 score than XGBoost [136] while

Car-Hacking Dataset : Multi-Class Confusion Matrix						
TARGET \ OUTPUT	Benign	DoS	Fuzzing	RPM-Spoof	Gear-Spoof	SUM
Benign	502584 81.036%	0 0.000%	7 0.001%	0 0.000%	0 0.000%	502591 99.999% 0.001%
DoS	0 0.000%	25760 4.153%	0 0.000%	0 0.000%	0 0.000%	25760 100.000% 0.000%
Fuzzing	4 0.001%	0 0.000%	32357 5.217%	0 0.000%	0 0.000%	32361 99.988% 0.012%
RPM-Spoof	0 0.000%	0 0.000%	0 0.000%	32385 5.222%	0 0.000%	32385 100.000% 0.000%
Gear-Spoof	0 0.000%	0 0.000%	0 0.000%	0 0.000%	27103 4.370%	27103 100.000% 0.000%
SUM	502588 99.999% 0.001%	25760 100.000% 0.000%	32364 99.978% 0.022%	32385 100.000% 0.000%	27103 100.000% 0.000%	620189 / 620200 99.998% 0.002%

Fig. 5.9 Confusion matrix capturing the classification results of the QLSTM-IDS for the Car Hacking Dataset. The figure also shows the success/failure rates for individual attacks as well as overall performance numbers.

being on par with the other two models in detecting flooding attacks. In the case of detecting fuzzing and malfunction attacks, the QLSTM-IDS improves on XGBoost [136] and Random Forest [138] by 0.4% & 0.07% (fuzzing), and by 0.04% & 0.06% (malfunction) respectively, while being nearly identical to the LSTM [137] model.

When tested on the **Soul** subset, the QLSTM-IDS shows similar improvements over the state-of-the-art methods. When detecting flooding attacks, the model achieves perfect detection with zero misclassification across the test set, which is on par with Random Forest, LSTM, and Transformer-based IDS, while improving on the XGBoost model by 5.3% (F1 score). For fuzzing attack detection, the model improved on the LSTM and Transformer models by 2.6% and 2.3% (F1 scores) respectively, while slightly falling behind XGBoost and Random Forest by 0.2% and 0.3% (F1 scores). Finally, in the case of malfunction attacks, the model outperforms all the other models with perfect detection with zero misclassification, improving on

Survival Analysis Dataset - Confusion Matrix (Sonata)					
TARGET \ OUTPUT	Benign	Flooding	Fuzzy	Malfunction	SUM
Benign	59492 79.323%	0 0.000%	1 0.001%	0 0.000%	59493 99.998% 0.002%
Flooding	0 0.000%	6735 8.980%	0 0.000%	0 0.000%	6735 100.000% 0.000%
Fuzzy	6 0.008%	0 0.000%	4774 6.365%	0 0.000%	4780 99.874% 0.126%
Malfunction	0 0.000%	0 0.000%	0 0.000%	3992 5.323%	3992 100.000% 0.000%
SUM	59498 99.990% 0.010%	6735 100.000% 0.000%	4775 99.979% 0.021%	3992 100.000% 0.000%	74993 / 75000 99.991% 0.009%

Fig. 5.10 Confusion matrix for QLSTM-IDS classification results on HYUNDAI YF Sonata 2010 Model attack set of SA dataset.

Survival Analysis Dataset - Confusion Matrix (Soul)					
TARGET \ OUTPUT	Benign	Flooding	Fuzzy	Malfunction	SUM
Benign	73511 81.679%	0 0.000%	0 0.000%	0 0.000%	73511 100.000% 0.000%
Flooding	0 0.000%	8705 9.672%	0 0.000%	0 0.000%	8705 100.000% 0.000%
Fuzzy	3 0.003%	0 0.000%	6277 6.974%	47 0.052%	6327 99.210% 0.790%
Malfunction	0 0.000%	0 0.000%	0 0.000%	1457 1.619%	1457 100.000% 0.000%
SUM	73514 99.996% 0.004%	8705 100.000% 0.000%	6277 100.000% 0.000%	1504 96.875% 3.125%	89950 / 90000 99.944% 0.056%

Fig. 5.11 Confusion matrix for QLSTM-IDS classification results on KIA compact SUV Soul 2015 Model attack set of SA dataset.

Survival Analysis Dataset - Confusion Matrix (Spark)					
TARGET \ OUTPUT	Benign	Flooding	Fuzzy	Malfunction	SUM
Benign	40405 80.810%	0 0.000%	123 0.246%	0 0.000%	40528 99.697% 0.303%
Flooding	0 0.000%	7011 14.022%	0 0.000%	0 0.000%	7011 100.000% 0.000%
Fuzzy	26 0.052%	0 0.000%	1454 2.908%	0 0.000%	1480 98.243% 1.757%
Malfunction	0 0.000%	0 0.000%	0 0.000%	981 1.962%	981 100.000% 0.000%
SUM	40431 99.936% 0.064%	7011 100.000% 0.000%	1577 92.200% 7.800%	981 100.000% 0.000%	49851 / 50000 99.702% 0.298%

Fig. 5.12 Confusion matrix for QLSTM-IDS classification results on CHEVROLET mini-compact vehicle Spark 2015 Model attack set of SA dataset.

XGBoost, Random Forest, LSTM, and Transformer models by 3.5%, 0.06%, 9.1%, and 0.2% (F1 scores), respectively.

A very similar performance from the model is observed when tested on the **Spark** subset, with a dip in performance observed in the case of the fuzzing attack. When detecting flooding and malfunction attacks, the model achieves perfect detection with zero misclassification across the test data, improving on or matching the performance of the competition. However, when detecting the fuzzing attack, the model achieves good precision but with degraded recall, F1 scores, and false negative rates, producing better results than only the XGBoost model. This degradation stems from the very low number of test data available in the Spark subset and specifically for the fuzzing attack on this vehicle. Additionally, it should be noted that all competing methods (except XGBoost) use a block of acquired CAN data as the input feature for detecting the attack, with adapted labels that flag the block as malicious; in contrast, the proposed model performs detection on a per-message basis and flags only the specific message as an attack or benign. The per-message classification can aid in maintaining a higher degree of functionality in the vehicle in the presence of an attack, and hence is a more viable solution for deploying IDS for intra-vehicular networks like CAN.

Table 5.3 Performance metric comparison of QLSTM-IDS against different models proposed for the Sonata set of the SA dataset.

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	100	90	94.74	-
Random Forest [138]	100	100	100	0
LSTM [137]	-	100	100	0
QLSTM-IDS	100	100	100	0

(a) Flooding Attack

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	99.98	99.08	99.53	-
Random Forest [138]	99.79	99.93	99.86	-
LSTM [137]	-	99.95	99.96	0.05
QLSTM-IDS	99.87	99.98	99.93	0.02

(b) Fuzzing Attack

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	99.92	100	99.96	-
Random Forest [138]	99.90	99.98	99.94	-
LSTM [137]	-	100	100	0
QLSTM-IDS	100	100	100	0

(c) Malfunction Attack

Additionally, in [152], the authors only report an average F1 score of 97.6% for the IDS model proposed in the paper, when averaged across all attacks and all vehicles. In comparison, QLSTM-IDS achieves average F1 scores of 99.97%, 99.86%, and 98.37% across the attack datasets for the Sonata, Soul, and Spark models, respectively, with an overall average of 99.4%. Overall, the proposed QLSTM-IDS demonstrates competitive performance against state-of-the-art IDS models across all tested scenarios, matching or besting state-of-the-art in many cases, while being close to the state-of-the-art when it fell short. The results reinforce the hypothesis on the effectiveness of the quantised LSTM-Dense model (QLSTM-IDS) in performing robust multi-class classification across diverse automotive attacks.

Table 5.4 Performance metric comparison of QLSTM-IDS against different models proposed for the Soul set of the SA dataset.

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	100	90	94.74	-
Random Forest [138]	100	100	100	0
LSTM [137]	-	100	100	0
Transformer [132]	100	100	100	-
QLSTM-IDS	100	100	100	0

(a) Flooding Attack

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	99.99	99.58	99.78	-
Random Forest [138]	99.79	99.93	99.86	-
LSTM [137]	-	94.69	97.01	5.31
Transformer [132]	96.84	97.84	97.33	-
QLSTM-IDS	99.21	100	99.60	0

(b) Fuzzing Attack

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	100	93.22	96.49	-
Random Forest [138]	99.90	99.98	99.94	-
LSTM [137]	-	96.97	90.87	3.03
Transformer [132]	99.65	99.95	99.80	-
QLSTM-IDS	100	100	100	0

(c) Malfunction Attack

5.4.2 Detection Latency

To determine if the combination of the proposed low-precision QLSTM model, its dataflow accelerator, and the direct integration with the network controller strategy can enable near-line-rate performance, the CAN controller’s internal buffers are pre-loaded with sample CAN messages from the dataset and determine the processing latency in hardware. In this chain, the elapsed time from the CAN controller passing the message to the Zynq processor (and hence to the IDS) to the processor reading the classification result from the IDS model is measured. The measurement is started

Table 5.5 Performance metric comparison of QLSTM-IDS against different models proposed for the Spark set of the SA dataset.

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	100	90	94.74	-
Random Forest [138]	100	100	100	0
LSTM [137]	-	100	100	0
Transformer [132]	100	100	100	-
QLSTM-IDS	100	100	100	0

(a) Flooding Attack

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	97.34	92	94.6	-
Random Forest [138]	99.79	99.93	99.86	-
LSTM [137]	-	98.23	98.31	1.77
Transformer [132]	98.85	99.39	99.12	-
QLSTM-IDS	98.24	92.20	95.13	7.8

(b) Fuzzing Attack

Model	Precision (%)	Recall (%)	F1 (%)	FNR (%)
XGBoost [136]	99.80	100	99.90	-
Random Forest [138]	99.90	99.98	99.94	-
LSTM [137]	-	100	100	0
Transformer [132]	99.17	99.78	99.47	-
QLSTM-IDS	100	100	100	0

(c) Malfunction Attack

when the controller's data interface to the processor (and IDS) asserts the data valid signal indicating that new frame data is available to be read. The input multiplexer for the pre-processing FIFO is set to receive data from the interface to capture the latency through the hardware chain. Alternatively, the AXI-DMA could have been used to load the CAN messages into the pre-processing FIFO from the PYNQ interface on the ARM cores and start the timer immediately before asserting the enable input on the IDS (measurement via software timers). In the second case, the input multiplexer for the pre-processing FIFO is set to receive data from the AXI-DMA (and by extension from software), with a delayed enable signal to capture

the latency through the hardware chain. As mentioned, the PYNQ OS image is used on the ARM cores; hence, both measurements factor in OS and driver overheads for reading the result from the IDS accelerator. Minimal variation is observed between the two measurement flows and the average latency across multiple runs is computed to minimise the impact of OS/software-level timing variations. The measurements are also performed on a per-message basis, mimicking the real data flow in an in-vehicle network.

Table 5.6 presents a comparative analysis of the proposed accelerator’s per-message processing latency against SOTA models implemented on GPUs, FPGAs, or microcontroller platforms. Note that in approaches in the literature where block-based processing is used, the reported latency does not consider the time taken to accumulate the required window of messages before processing can be commenced, and hence presents a skewed view of the latency in a message (packet) network context. It can be observed that the QLSTM-IDS accelerator and integration achieve an impressive end-to-end latency of 0.25 ms per message, which is $2.29\times$ faster than the other per-message processing schemes such as MTH-IDS [23]. The QMLP and CQMLP IDS models presented in Chapter 3 benefit from the reduced computational complexity of the MLP model, and can process messages with $1.04\times$ and $2.27\times$ lower latency, respectively. However, the MLP models use sliding window-based processing to capture the time-series relations in CAN messages, while the threshold-based detection scheme flags the entire window as malicious when a threshold is met, which is much less efficient than the QLSTM model. The 0.25 ms per message processing latency enables the proposed IDS to handle over 4,000 messages per second, a throughput that supports robust real-time intrusion detection in most of the safety-critical deployment scenarios that use the CAN network. It is observed that while the accelerator operates at a higher frequency of 150 MHz for this testing, there is considerable latency incurred by the native PYNQ read/write APIs. With a bare-metal or RTOS implementation, it is anticipated that the per-message latency would be further improved by reducing the data movement overheads.

5.4.3 Energy & Resource Utilisation

To evaluate the energy consumption of the IDS, the `pynq-rails` feature from the PYNQ-PMBus package is used to monitor and isolate the power rails of the processor subsystem (ARM-cores) and programmable logic (logic region). To measure the energy cost of the IDS, any power rails that power the logic region of the Zynq are

Table 5.6 Detection latency comparison of the proposed QLSTM-IDS against other state-of-the-art IDSs reported in the literature.

Models	Latency	Frames	Platform
GRU [94]	890 ms	5000 CAN frames	Jetson Xavier NX
MLIDS [31]	275 ms	per CAN frame	GTX Titan X
NovelADS [93]	128.7 ms	100 CAN frames	Jetson Nano
TAN-IDS [132]	11.6 ms	128 CAN frames	-
DCNN [18]	5 ms	29 CAN frames	Tesla K80
TCAN-IDS [92]	3.4 ms	64 CAN frames	Jetson AGX
MTH-IDS [23]	0.574 ms	per CAN frame	Raspberry Pi 3
QMLP-IDS [133]	0.24 ms	per CAN frames	Zynq Ultrascale+
BIDS [149]	0.17 ms	48 CAN frames	Zynq
CQMLP-IDS [150]	0.11 ms	per CAN frames	Zynq Ultrascale+
QLSTM-IDS	0.25 ms	per CAN frame	Zynq Ultrascale+

monitored while triggering the IDS to perform inference on a test dataset. The measurements are averaged over 10000 messages and inference operations, with each new message arriving at the peak rate (4000 messages per second) determined from the latency measurement. It is observed that the IDS implementation incurs 0.8 mJ on average per inference at the peak input rate. While this consumption is marginally higher than the energy consumed by previous quantised MLP models, the proposed model represents a better trade-off to achieve better IDS functionality, through multi-class attack classification with a higher classification accuracy and better generalisability across multiple attack modes and datasets.

The resource utilisation of the IDS accelerator and the CAN network controller (CAN-NC) are further assessed, with metrics gathered post-implementation via Vitis-HLS. As detailed in Table 5.7, the accelerator uses less than 20% of LUTs and under 10% of flip-flops (FFs), leaving substantial resources available for additional accelerators or applications on the FPGA. This efficient resource utilisation highlights the design’s scalability and flexibility, allowing other concurrent workloads to be offloaded from the ECU to hardware blocks on the logic region, without affecting IDS functionality.

5.4.4 Discussion: QNNs proposed as IDSs

As shown in the results section, several quantised IDS models have recently been proposed in the research literature to reduce model complexity and energy consumption. This subsection examines their accuracy metrics and deployment statistics in

Table 5.7 The resource consumption overheads of QLSTM-IDS and the CAN Controller on the PL region of the Zynq XCZU7EV SoC.

Function	LUTs (%)	FFs (%)	LUTRAM (%)	BRAMs (%)	DSPs (%)
QLSTM-IDS	40902 (17.75)	11234 (2.44)	4669 (4.59)	23.5 (7.53)	12 (0.69)
CAN-NC	887 (0.38)	625 (0.14)	18 (0.02)	-	-

comparison to the proposed quantised LSTM-based IDS (QLSTM-IDS), which are shown in Table 5.8. In [149], the authors present a two-stage IDS consisting of a Binarised Neural Network (BNN) and a GAN. Their model uses a sequence of CAN IDs with a window length of 48 messages as input features. However, using only CAN IDs as input features limits the model’s ability to detect attacks that inject malicious payloads without affecting the IDs. Additionally, the 1-bit quantised IDS has a higher misclassification rate compared to the proposed QLSTM-IDS. The QMLP design in chapter 3 uses two separate 8-bit quantised MLP models to detect DoS, Fuzzing, and Spoofing (RPM and Gear) attacks. The key idea was to optimise the learned features for similar attack vectors, thus improving the individual detector’s performance while reducing the model complexity. It is observed that the proposed QLSTM-IDS achieves a lower misclassification rate compared to the tailored models while reducing the energy consumption per inference by over $2 \times$. Similarly, the multi-class classification performance of the CQMLP model in chapter 3 was achieved using a conservative labelling scheme to improve model convergence and to achieve high accuracy, using a sliding window approach over input (and previous) CAN messages. With the final implementation, the CQMLP only manages to achieve a slightly higher misclassification rate than the proposed QLSTM-IDS, while the window-based labels and thresholding scheme can lead to dropped benign messages at runtime. In [153], authors propose a fully binarised neural network as an IDS, with a very low detection latency of 0.04 ms. However, the authors validate the models’ performance on a proprietary custom dataset, with the results showing lower accuracy metrics ($\approx 90.5\%$) for random ID, replay, spoofing, and other hybrid attacks in this dataset.

Compared to other implementations in Table 5.8, QLSTM-IDS achieves the lowest misclassification rate (11/620,200), outperforming QMLP, BIDS, and CQMLP by a significant margin. It uses moderate hardware resources – 17.75% LUTs and 2.44% FFs – lower than QMLP and BIDS while maintaining minimal BRAM usage (0.82

Table 5.8 This table compares the proposed QLSTM-IDS with other QNN models from the research literature, evaluating them in terms of resource utilisation, energy consumption, and overall misclassification rates as reported on the car hacking dataset.

Model	LUTs	FFs	BRAM (Mb)	Energy (mJ)	Misclassifications
QMLP [133]	56,733 (24.6%)	72,147 (15.6%)	3.06	1.96	75/200,000
CQMLP [150]	3999 (1.74%)	4524 (0.98%)	0.84	0.23	147/180,000
BIDS [149]	23,256 (10.09%)	37,943 (8.2%)	4.07	0.35	70/187,062
QLSTM-IDS	40,902 (17.75%)	11,234 (2.44%)	0.82	0.8	11/620,200

Mb). Although its energy consumption (0.8 mJ) is higher than CQMLP and BIDS, it remains more efficient than QMLP. Each inference takes only 0.25 ms, making it well-suited for real-time detection. Overall, QLSTM-IDS offers the best balance between accuracy, resource utilisation, and deployability among the evaluated models.

5.5 Summary

In this chapter, a multi-class classification model that intrinsically captures the time-series dependencies of in-vehicle messages was developed using a quantised LSTM-Dense network. An integration approach that moves the IDS closer to the network interface is also discussed, using the Zynq Ultrascale+ XCZU7EV device as a template ECU. The model is evaluated on two open-source datasets, the Car Hacking Dataset, and the Survival Analysis Dataset to train the model and to evaluate its performance in detecting multiple attack vectors. To address practical deployment constraints, mixed precision quantisation (weights and activations; obtained through a DSE) is used to reduce the computational requirements of the LSTM-Dense model without impacting the classification accuracy. A custom export flow is designed for converting the quantised model into the intermediate representation and further developed custom transformations that map the compute nodes in the intermediate representations to existing hardware library blocks in the FINN compiler flow, to generate the dataflow accelerator (QLSTM) of the model. The results show that the proposed quantised LSTM model achieves exceptional average accuracies of 99.998% and 99.90% across the two datasets, and improves upon or matches the state-of-the-art models in almost all cases while consuming only 0.8 mJ of energy per inference action. The model also showed the capability to process close to 4,000 messages per second, enabling real-time detection in most of the critical CAN

deployment scenarios. The proposed model achieves the best trade-offs between robustness, detection accuracy, energy efficiency, and feasibility for deployment in an embedded environment, making it ideally suited for augmenting the security of automotive CAN systems. An important observation is the reduction of data movement latency which was achieved by moving the IDS closer to the network interface using the bridged integration. The training and implementation flows are openly released to the community with the hope that it will inspire more research into efficient time-series models for IDS and other capabilities.

5.6 Related Publications

- S. Khandelwal, J. Konig, T. Pressuer, M. Blott, S. Shreejith, "FINN-GL: Generalized Mixed-Precision Extensions for FPGA-Accelerated LSTMs", International Conference on Field-Programmable Logic and Applications (**FPL**), Leiden, The Netherlands, 2025
- This application part of the chapter is currently under review in the ACM TECS journal.

Chapter 6

SecCAN: CAN Controllers with Embedded Intrusion Detection

6.1 Introduction & Contributions

In the previous chapters, ML-based IDS models that are compressed and optimised for efficient deployment were discussed. Despite the reduced complexity, tightly coupled hardware accelerators – either in the form of custom logic or specialised compute units like GPUs, were relied on to meet the latency requirements, as shown in previous chapters and competing research publications. In such architectures, IDS processing typically begins only after the entire message is received by the ECU and forwarded to the IDS engine. This integration strategy is consistent across most works in the literature, where the focus has largely been on developing accurate detection models, with relatively little attention paid to the architectural aspects of IDS integration that could potentially lead to early detection.

Figure 6.1 captures the different IDS integration strategies proposed in the literature. Case 1 in figure 6.1 shows the IDS integrated as a software task on an existing ECU, while case 2 shows a dedicated IDS accelerator where the IDS could be deployed as the lone software task on an ECU, GPU (edge device or a standard GPU) or a microcontroller platform like Raspberry Pi [154]. Case 3 shows a coupled accelerator, where the ECU offloads IDS to the accelerator on the same SoC once the CAN controller receives the packet. With all the above cases, the CAN message has to be entirely received by the controller without any error flags and subsequently read by the coupled ECU / processing element for detection to begin. In an ideal scenario, the malicious content detection could overlap with message reception and

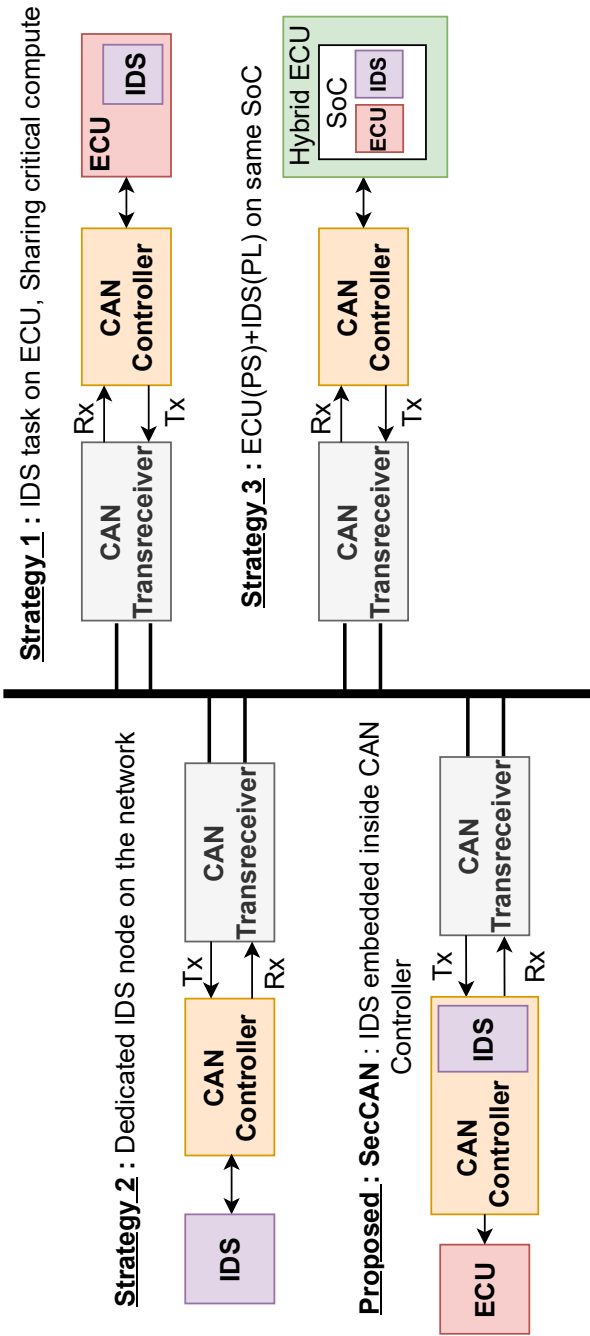


Fig. 6.1 The figure illustrates conventional integration and placement strategies for CAN intrusion detection systems presented in the literature, alongside the proposed case for embedding IDS within the controller.

decoding of the CAN messages in the controller to take advantage of the serial bus

and redundant sampling employed by the protocol. In chapter 5, a hybrid integration strategy was envisaged that could reduce the data movement latency by bridging the interface between the microcontroller (ARM cores in this case) and the CAN controller.

In this chapter, this is further explored through an extended CAN controller, SecCAN, which integrates an IDS accelerator into its receive side datapath. The IDS accelerator reads the IDs and data bytes from within the byte-decoding logic in the receive path to generate the input feature vector for the IDS model. The integration is completely isolated from standard message flow within the controller and hence does not affect the normal functionality of the controller or the connected ECU. For the IDS, a lightweight 4-bit quantised multi-layer perceptron (QMLP) model is employed which is trained on the open car hacking and survival analysis datasets to detect DoS/flooding, fuzzing, and malfunction attacks. The dataflow-optimised IDS architecture combined with the integration within the controller allows the SecCAN controller to detect and tag incoming CAN messages before the transmission window is completed, and optionally trigger error frames to alert downstream ECUs. The IDS performance of the SecCAN controller is evaluated by replaying messages from the open Car hacking & survival analysis dataset on the test platform. The key contributions of this chapter are as follows:

- A novel CAN controller architecture (SecCAN) is designed that seamlessly integrates IDS capabilities into the datapath of the controller without affecting standard message flow, thus being fully transparent to the ECU.
- A lightweight 4-bit QMLP model was developed and dataflow optimised to generate the low-latency quantised IDS (Q-IDS) accelerator integrated into SecCAN. The model was trained and tested on both survival analysis and car hacking datasets. The specific model achieved state-of-the-art binary classification accuracy of 99.993% and 99.966% (average) across multiple attacks.
- The SecCAN controller is implemented on an AMD Zynq XCZU7EV FPGA to characterise its performance and to determine the latency, area, and energy benefits/overheads. The tests show that SecCAN enables line-rate IDS capability with zero latency and zero software overheads incurred by the ECU even at the highest CAN data rates. Compared to dedicated IDS accelerators such as on a Jetson Xavier GPU, SecCAN is significantly more energy-efficient ($34\times$) with the IDS implementation in SecCAN incurring much lower latency ($10.9\times$).

The datapath changes in SecCAN allow the IDS to be fully offloaded from the ECU, with clever overlapping of IDS with the message reception from the CAN bus. This makes IDS fully transparent to the ECU, effectively incurring zero latency for the IDS and with zero change to existing software tasks, a key difference to competing ML-based IDS methods. SecCAN implementation is made publicly available to foster further research into smart network interfaces at <https://github.com/RCSL-TCD/SecCAN>.

The remainder of this chapter is organised as follows: Section 6.2 discusses different IDS integration strategies presented in the literature; Section 6.3 presents the proposed IDS integration strategy with the CAN controller; Section 6.4 presents the results of the integration and the chapter is concluded in Section 6.5.

6.2 Integration Strategies

Figure 6.1 illustrates various IDS integration strategies that have been explored in existing research literature for CAN. These strategies represent different architectural approaches to embedding IDS functionality within automotive networks, each with its own advantages and trade-offs. The four primary IDS integration strategies are as follows:

- **IDS task on ECU:** In this approach, the IDS is implemented as a software task running on an existing ECU. This method leverages the computational resources of the ECU to monitor network traffic and detect potential intrusions. While this approach minimises additional hardware requirements, it may impose a computational burden on the ECU, potentially affecting its real-time performance for other critical automotive functions [155, 156].
- **Dedicated IDS node on the network:** This strategy involves deploying a standalone IDS node within the vehicle's network. The dedicated IDS node is responsible for monitoring communication traffic across multiple ECUs, providing a centralized security mechanism. Although this approach reduces the computational load on individual ECUs, it introduces additional hardware costs/weight issues and may become a bottleneck in resource-constrained vehicular environments [157–159].
- **ECU + IDS on same SoC:** In this configuration, the IDS is tightly integrated with an ECU by deploying both functionalities on the same SoC. This

approach allows for optimised resource sharing between the ECU and IDS while maintaining low latency and power efficiency. This integration has been extensively used in previous chapters.

- **Distributed IDS Across Multiple ECUs:** A distributed IDS architecture involves deploying IDS functionalities across multiple ECUs within the network. Each ECU runs a lightweight IDS module that contributes to a collaborative detection mechanism. This decentralised approach enhances scalability and resilience by distributing the computational workload and detection responsibilities. However, it requires efficient coordination and synchronization between ECUs to ensure comprehensive threat detection without excessive overhead [160].

Each of these IDS integration strategies offers unique benefits and challenges, depending on factors such as computational constraints, latency requirements, and security needs. However, in all existing IDS integration approaches, the detection process only begins once the ECU has fully received and processed the network message from the controller. This delayed detection mechanism can introduce latency in identifying malicious activity, potentially allowing an attack to propagate further before mitigation measures are triggered. Additionally, since the IDS operates after message reception, it may struggle to meet real-time processing constraints, particularly in high-speed automotive networks with stringent timing requirements.

In this chapter, a datapath extension is designed within an otherwise standard CAN controller which embeds the intrusion detection model. This integration allows the IDS model to directly read byte-decoded data from the CAN bus as the message is being received. A tag-bit mechanism is also defined whereby the message is attached with a flag bit that indicates its malicious state when the message is ready to be read by the ECU. This minimises flagging overheads for each message as it forms part of the standard read message sequence in the ECU's software task. The ECUs can then use this tag-bit to make informed decisions about message handling, such as prioritizing, filtering, or isolating suspicious messages before full processing.

By integrating the IDS within the CAN controller's datapath, this approach offloads the capability to the network interface and masks IDS latency within the controller, reducing the risk of delayed threat response. This strategy not only improves real-time security monitoring but also minimizes the computational burden on individual ECUs, making it a scalable and efficient solution for automotive networks.

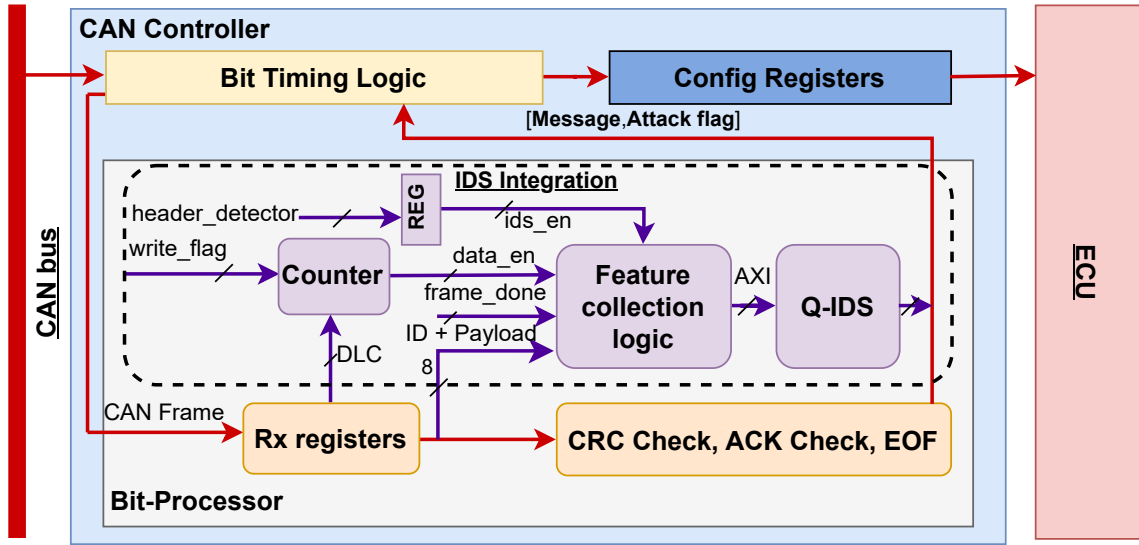


Fig. 6.2 The figure illustrates the integration of the IDS within the CAN controller. The red arrows show the standard datapath within the controller with the purple arrows indicating the augmented path for IDS.

6.3 SecCAN: IDS integration

6.3.1 Extending Controller Datapath for IDS

The CAN controller implements the CAN protocol at the logic level using functional blocks that perform bit-level processing, data packaging/framing, error detection, and optional message filtering functions. For this work, an open-source register transfer level (RTL) implementation of the CAN controller described in Verilog is utilised as the baseline CAN controller [161]. The functional blocks of the protocol are implemented in the baseline CAN controller through the following modules: `config_registers`, `bit_timing_logic` and `bit_processing`.

The `config_registers` module manages the interface to the ECU and implements a set of registers to capture protocol configuration, status signals, and the transmission and receive frame buffers over an AXI interface. At startup, the ECU configures the protocol registers, which control operating parameters such as bit-level timing (sampling windows, clock prescaler configuration), interrupt configuration, and message filtering for the receive path, among others. Once configured, the ECU writes up to 8 bytes of data to be sent as a CAN frame into the transmit registers,

and the controller attempts to transmit the payload based on its arbitration priority. The status of transmission is subsequently updated on the status register. When a CAN frame is received from the bus, the message filter configuration determines if it needs to be passed to the ECU or dropped by the controller. If the frame is to be passed to the ECU, the CAN frame is written to the receive registers and generates an interrupt signal (if not masked) to the ECU, causing the ECU to read the received CAN frame.

The `config_registers` module implements the configuration registers and their mapping to other blocks. The register space implements status, error tracking, and command/control registers. The `bit_timing_logic` module implements the bit-level timing and synchronization functions of the protocol and interfaces with the physical CAN bus. The `bit_processing` module processes the encoding/decoding of bits to/from the CAN bus and implements the transmit, receive, and error handling functions. A 4-bit quantised MLP is integrated as the IDS within the bit processing module of the CAN IP, as shown in figure 6.2. The QNN IP is generated using AMD's FINN toolchain with an AXI streaming interface for receiving input CAN features.

The IDS operation begins when a known header is detected on the bus, triggering the `header_detector` signal. This marks the start of message transmission asserting `ids_en`, at which point the IDS is ready. A counter is employed to track the number of data bytes written to the receive FIFO, using the internal `write_flag` signal. When all bytes of a CAN frame along with the message identifier have been stored in the FIFO (comparing counter value against data length code {DLC}), the `data_en` signal of the IDS IP is asserted to initiate a message write to it. Over the next 10 clock cycles, the input features used to process each message (CAN ID + Payload) are transferred to the IDS IP. Note that if the message length is less than the maximum 8 bytes, additional bytes are padded with zeros to ensure compatibility with the IDS, which requires 10 input features. This padding logic is integrated into the IDS input preparation stage in the `bit_processing` module, allowing the controller to use the dataset directly without pre-processing. Once the data transfer is complete, the IDS processes the frame for potential attacks. The IDS completion is indicated by the assertion of the `ids_output_ready` signal. After processing, the IDS output is transferred from the `bit_processing` module to the `top` module, where custom multiplexing logic is added to the `data_out` interface. This enhancement appends the IDS output to the last bit of each decoded message, marking it as either benign or an attack. Finally, the modified message flows through the `bit_timing_logic`

and `config_registers` modules before being transmitted to the ECU. Figure 6.3 illustrates the SecCAN operation using a waveform, from the start of transmission to the frame completion state. Optionally, SecCAN could be extended to an IPS function, which can automatically generate a high-priority error frame when an attack is detected, causing other ECUs to discard the flagged message.

6.3.2 Model & Dataset for training and testing

To arrive at the final QNN model architecture, different model configurations with varying complexity (number of layers and number of neurons in each layer, quantisation levels) were explored to find a model that offers high inference accuracy at minimal complexity. CAN ID + Payload information from each CAN frame is used as the input feature for the model to perform binary classification. A 4-bit (weights/activations) quantised MLP was chosen configuration as the IDS, which provided the best validation accuracy during the training process with `brevitas`. The final model was trained for 200 epochs with the `adam` optimizer and `lr` set to 0.0001. The model consists of 3 `QuantLinear` layers (input, one hidden & output ; `Linear` layer equivalent in `pytorch`) with {64, 32, 1} neurons at each layer, followed by `QuantRelu` activations. Batch normalisation and dropout layers were used between the `Quantlinear` layers to prevent overfitting during the training phase. The output layer uses the `sigmoid` activation function to estimate the probability of whether the input message is an attack or an expected bus message. The model was trained and tested on both the car hacking and survival analysis datasets. Each dataset was split into 75:15:10 proportions for training, validation, and testing respectively.

6.4 Evaluation & Results

The hardware configuration of the model is generated using AMD Vivado 2022.2, with the XCZU7EV device on the ZCU104 development board as the FPGA platform. The CAN controller is set to operate at a maximum bitrate of 1 Mbps, with a CAN clock frequency of 16 MHz. The 4b-QMLP IDS model's accuracy in detecting active injection attacks is evaluated and compared to state-of-the-art approaches in the literature. The 4b-QMLP model uses the same 16 MHz to avoid clock domain crossing; however, it can be synthesized for a much higher clock frequency (100 MHz) if required, trading off reduced resource and energy consumption (as discussed in sec. 6.4.3). Additionally, the inference latency in hardware is measured and the

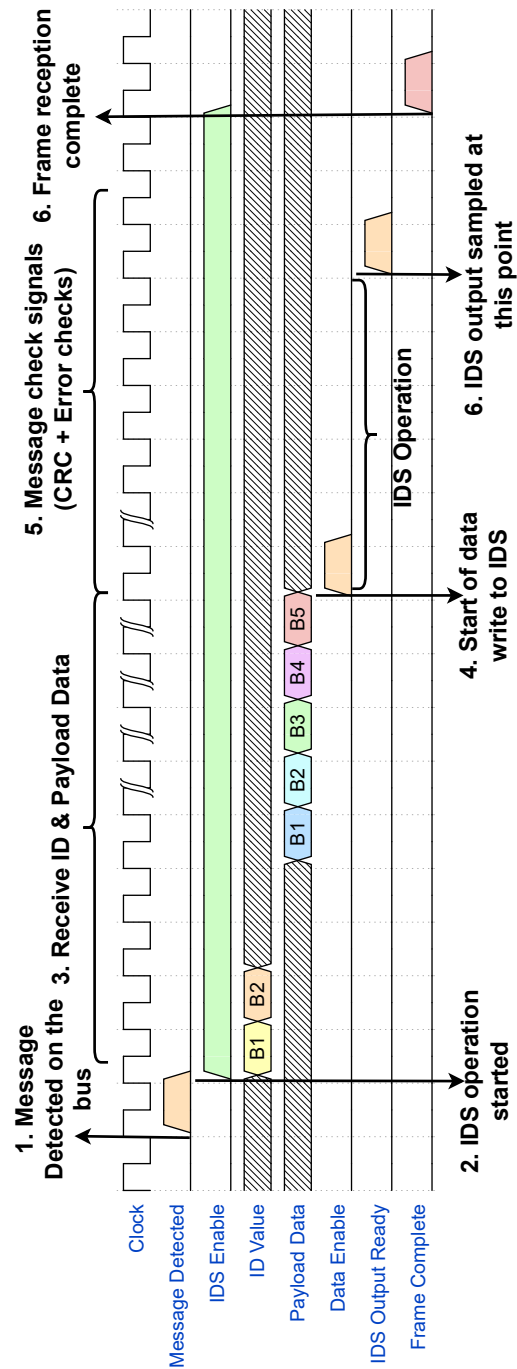


Fig. 6.3 The waveform shows the signalling within the SecCAN controller for a 5-byte CAN message from the bus. The IDS operation overlaps with the protocol checks on the bus and is completed before the frame is ready to be read by the ECU.

resource and energy consumption trade-offs are analysed. For measurements, the attack messages are replayed either directly from the ARM processor core or from a BRAM that replays CAN traffic data (for quantifying latency and power).

6.4.1 Accuracy

The accuracy of the 4b-QMLP model is quantified in terms of precision, recall & F1 scores and compared against state-of-the-art works in the literature, and across multiple datasets. In the case of the car hacking dataset, the test set comprises 75,000 messages, combined from the DoS (benign: 58,769, attack: 16,231) & fuzzing (benign: 59,686 & attack: 15,314) attack datasets. The model achieves a binary classification accuracy of 99.993% (11 misclassifications for 150,000 test messages), For the survival analysis dataset, the test set comprises flooding (benign: 18,265, attack: 6735), fuzzing (benign: 20,220, attack: 4780), and malfunction (benign: 21,008, attack: 3992) attacks, respectively. For this dataset, the model achieves a classification accuracy of 99.966% (25 misclassifications for 75,000 test messages). Table 6.1 presents a comparison of the model with others proposed in the literature for the two attack datasets, highlighting that the proposed approach matches or surpasses the performance of competing methods.

6.4.2 Detection Latency

The CAN controller is set to operate at 1 Mbps operating speed as shown in the equation,

$$\text{Bit Rate} = \frac{\text{Clock Frequency}}{2 \cdot (\text{BRP} + 1) \cdot (\text{TSEG1} + \text{TSEG2} + 1)}$$

where,

- **BRP (Baud Rate Prescaler):** Determines the division of the input clock.
- **TSEG1 (Time Segment 1):** Controls the propagation segment and part of the phase buffer segment.
- **TSEG2 (Time Segment 2):** Controls the second part of the phase buffer segment.
- **SJW (Synchronization Jump Width):** Used for re-synchronization in case of clock drifts.

Table 6.1 Comparison of detection accuracy (%) of the 4b-IDS solution on the car hacking (CH) and the survival analysis (SA) datasets.

Attack	Model	Precision	Recall	F1	FNR
DoS (CH)	DCNN [18]	100	99.89	99.95	0.13%
	NovelADS [93]	99.97	99.91	99.94	-
	TCAN-IDS [92]	100	99.97	99.98	-
	GRU [94]	99.93	99.91	99.92	-
	iForest [154]	95.07	99.93	97.44	-
	4b-QMLP in SecCAN	99.99	99.98	99.98	0.02%
Fuzzing (CH)	DCNN [18]	99.95	99.65	99.80	0.5%
	NovelADS [93]	99.99	100	100	-
	TCAN-IDS [92]	99.96	99.89	99.22	-
	GRU [94]	99.32	99.13	99.22	-
	iForest [154]	95.07	99.93	97.44	-
	4b-QMLP in SecCAN	99.99	99.97	99.98	0.03%
Flooding (SA)	XGBoost [162]	100	90	94.74	-
	Random Forest [138]	100	100	100	0%
	LSTM [163]	-	100	100	0%
	4b-QMLP in SecCAN	100	100	100	0 %
Fuzzing (SA)	XGBoost [162]	99.98	99.08	99.53	-
	Random Forest [138]	99.79	99.93	99.86	-
	LSTM [163]	-	99.95	99.96	0.05%
	4b-QMLP in SecCAN	99.98	99.50	99.74	0.5%
Malfunction (SA)	XGBoost [162]	99.92	100	99.96	-
	Random Forest [138]	99.90	99.98	99.94	-
	LSTM [163]	-	100	100	0%
	4b-QMLP in SecCAN	100	100	100	0%

For real-time message tagging, the IDS must complete the analysis of each message before the reception window is completed (end of error flags in CAN protocol). For a maximal-length CAN message, the reception window T_{max} is given by eq 6.1, where T_{frame_done} & $T_{header_detected}$ marks the time at completion of frame reception and valid header reception respectively.

$$T_{max} = T_{frame_done} - T_{header_detected} \quad (6.1)$$

Since the IDS uses header and payload as input features, the analysis can only start once all data bytes are decoded from the bus (indicated by $data_en$ signal). For

CAN bus operating at 1 Mbps, this time window can be determined as 37.376 μ s; hence, for real-time detection, the maximum latency for IDS (T_{IDS}) should be:

$$T_{IDS} = T_{frame_done} - T_{data_en} < 37.376\mu s \quad (6.2)$$

Eq 6.2 holds for any data-length at 1 Mbps CAN bus configuration. From the evaluation (both from simulation and hardware measurements), a detection latency of 36.5 μ s is observed at 16 MHz clock, allowing the SecCAN to tag each message with an attack or benign flag within the reception window. Further, a $10.9\times$ reduction in latency is observed for our unrolled 4b-QMLP compared to an 8-bit implementation of the model on a Jetson Xavier, even with reception and message transfer times excluded.

6.4.3 Resource & Energy Consumption

The resource consumption numbers of the SecCAN controller are quantified and compared to the standard CAN controller in table 6.2. It is observed that the IDS model contributes $\approx 29.5\%$ additional LUTs, $\approx 0.45\%$ additional FFs, and $\approx 60.5\%$ LUTRAMs over the standard CAN controller. The higher LUT/LUTRAM usage results from a fully unrolled implementation of the model, which allows the model to achieve line-rate analysis at the same clock rate of the controller. A higher operating clock frequency can be used to selectively unroll the model to reduce resource consumption, at the cost of higher energy consumption and interface complexity (metastability). The per-message energy consumption of the IDS is measured by monitoring the power rails on the ZCU104 board during execution of the IDS. The model consumes only 73.7 μ J per message on average, a $3.1\times$ improvement over the 2-bit quantised coupled accelerator IDS presented in Chapter 3, achieved primarily through lower operating speed and the Q-IDS IP optimisations. When compared to software IDS, SecCAN achieves a $34.1\times$ reduction in energy consumption over an 8-bit variant of our model on a Jetson Xavier GPU. Compared to other works that have reported energy metrics, SecCAN incurs $24.1\times$ and $28.4\times$ lower energy per inference over an LSTM-based IDS on Jetson Xavier and the convolutional autoencoder-based IDS on a Zynq XCZU7EV FPGA respectively presented in Chapter 4 respectively. [94]. Table 6.3 presents a comparative analysis of the energy consumption of SecCAN against other state-of-the-art IDSs reported in the research literature.

Table 6.2 IDS resource utilisation for the standard CAN controller & the proposed SecCAN controller.

	LUTs (%)	FFs (%)	BRAMs (%)	LUTRAMs (%)
CAN-NC	887 (0.38%)	625 (0.14%)	0 (0)	18 (0.02%)
4b-QMLP	67902 (29.47%)	2007 (0.44%)	0.5 (0.16%)	61482 (60.42%)
SecCAN	68888 (29.90%)	2737 (0.59%)	0.5 (0.16%)	61500 (60.44%)

Table 6.3 Comparison of SecCAN’s energy consumption per inference with other IDS approaches proposed in the literature.

Model	Platform	Energy consumption
GRU [94]	Jetson Xavier NX	1.77 <i>mJ</i>
QCAE [164]	Zynq Ultrascale+	2.09 <i>mJ</i>
MTH-IDS [23]	Raspberry Pi 3	1.29 <i>mJ</i>
iForest [154]	Raspberry Pi 4	390.6 μ J
SecCAN	Zynq Ultrascale+	73.7 μ J

6.5 Summary

In this chapter, a secure CAN controller architecture is presented that integrates a lightweight machine learning model as an IDS accelerator within the CAN controller (SecCAN controller) to detect the onset of intrusions from CAN traffic flow. By integrating the IDS within the controller, the proposed SecCAN controller can classify benign/attack messages before the reception window is completed—an approach that has not been explored before to the best of our knowledge. This flow allows safety mechanisms and intrusion prevention systems to be triggered as soon as the ECU receives a message, unlike conventional integration approaches where the IDS processing can begin only after the ECU has fully received the message from the CAN controller. The embedded IDS consumes only 73.7 μ J per classification of each message while achieving 99.993% & 99.966% detection accuracy across multiple attack vectors and datasets, with a low resource overhead of < 30% LUT & < 1% FF on an AMD Zynq XCZU7EV. The SecCAN architecture is scalable and can pave the way towards smarter CAN controllers with integrated IDS solutions in existing and future automotive networks.

6.6 Related Publications

- S. Khandelwal and S. Shreejith, "SecCAN: An Extended CAN Controller With Embedded Intrusion Detection," in **IEEE Embedded Systems Letters**, May 2025. doi: 10.1109/LES.2025.3572858.

Chapter 7

Summary & Future Research

In this thesis, the deployment challenges of Intrusion Detection Systems in automotive networks have been addressed, particularly within Controller Area Networks. The need for robust security solutions in modern vehicles is evident, but their real-world deployment is constrained by factors such as detection latency, resource consumption, and energy utilisation. To address these gaps, various IDS architectures and their integration into an ECU have been systematically evaluated, focusing on quantised neural networks for efficient and accurate intrusion detection.

7.1 Summary of Contributions

7.1.1 Motivation and Initial Exploration

The thesis begins by establishing the necessity for deployment evaluation metrics that go beyond detection accuracy, incorporating efficiency-focused constraints such as inference latency, resource consumption on the platform, and the energy footprint of the implementation. Through initial experiments, it was observed that quantised neural networks could significantly improve inference efficiency without sacrificing accuracy, making them a promising solution for real-time automotive IDS. Further, these networks could be seamlessly integrated in isolation on hybrid FPGA-based ECUs which makes them particularly suitable for deployment.

7.2 CNN-Based IDSs and Their Limitations

The baseline approach utilised quantised Convolutional Neural Networks as IDSs, building on the existing state-of-the-art research. These models achieved high

accuracy and low processing latency when trained using CAN IDs as input features. Deploying these models on DPUs (utilising the Vitis-AI toolchain) enabled a seamless integration architecture, facilitating efficient execution and tighter coupling with existing functions on the ECU. However, it was found that CNNs struggled when CAN IDs and message payloads were used as the input feature, and required large blocks of messages for inference rather than processing messages individually. To develop generalisable IDSs capable of detecting diverse attack vectors across different vehicles, it is essential to incorporate message payloads alongside CAN identifiers, as they contain critical semantic information necessary for distinguishing malicious behaviour from benign communication. This limitation made them less suitable for real-time IDS deployment in CAN networks utilising the entire message (CAN ID + payload) as input features.

7.3 MLP-Based IDSs for Improved Efficiency

To address the payload processing and input block size constraints, Multi-Layer Perceptrons were explored, which are well-suited for classification tasks on time-series data while being computationally efficient. They were explored at multiple quantisation levels which were arrived at through design space exploration:

- **8-bit weights/activations** – a suitably quantised model that achieves high accuracy when trained with both CAN IDs and payloads.
- **2-bit weights/activations** – an extremely quantised model that significantly reduces latency while maintaining effectiveness.

By leveraging conservative input processing techniques, the 2-bit model allowed multi-attack classification at extremely low message processing latency, enabling line-rate intrusion detection. This model demonstrated high resource efficiency and low-energy message processing. Additionally, a fully custom dataflow accelerator design was generated using AMD's FINN flow for seamless integration with ECU functionalities without disrupting critical operations.

7.4 Unsupervised Learning for Zero-Day Attack Detection

While the previous approaches (using supervised learning-based quantised CNNs and MLPs) were effective at detecting known attacks and signatures, a system was developed that could identify zero-day attacks. In this scenario, the IDS system could detect abnormal message flows or intrusion signatures that were not present in the training set. To achieve this, an unsupervised learning-based convolutional autoencoder model was designed that was trained exclusively on normal message flow within the vehicle (using only normal messages from the dataset). During testing, this model successfully identified previously unseen attack messages with an accuracy exceeding 99%, demonstrating its capability for real-time zero-day attack detection. This model was deployed using the DPU provided by Vitis-AI, and the custom integration architecture enabled efficient block-wise message processing with overlapping message reads, thereby ensuring that each message is analysed in real-time for prompt zero-day attack detection.

7.5 Capturing time-series signature using LSTMs

Despite the success of quantised supervised learning-based MLP as IDSs and unsupervised learning-based autoencoder models, a critical concern was the requirement of a window of messages to capture temporal characteristics of the normal CAN data exchange. This limitation led to the exploration of Long Short-Term Memory networks, which are better suited for learning time-series features. Though LSTMs were an ideal choice for modelling the sequential characteristics of in-vehicle message exchanges, their recurrent nature was sub-optimal for a dataflow accelerator design on an FPGA. However, given that feed-forward models could not fully meet real-time classification needs (once window acquisition times were factored in), quantised LSTMs were explored as the ideal-case supervised learning-based IDS design.

An 8-bit weight and 6-bit activation quantised LSTM-based IDS model was developed, which enabled per-message attack classification across multiple datasets while maintaining low energy and resource utilisation. A custom extension to the FINN mapping flow was developed that could generate an efficient dataflow accelerator model for the mixed precision LSTM model. A hybrid integration model was also investigated, whereby the model bridged the interface between the CAN

controller and the ECU function, to mitigate software data movement overheads. The combination of a quantised LSTM model, custom dataflow accelerator and bridged integration allowed this IDS solution to balance real-time processing requirements (0.25 ms per inference) with high detection performance ($> 99.99\%$ classification accuracy) and energy efficiency (0.8 mJ per inference).

Deploying these models as accelerators (tightly coupled or bridged) on a hybrid ECU (on AMD's Zynq FPGA platforms) allows the IDS capability to be constantly updated through over-the-air weight and/or architecture updates (via updated bitstreams), enabling continuous learning and adaptation as new attack scenarios emerge.

7.6 Integration of IDS in CAN Network Controllers

After developing effective supervised and unsupervised ML-based IDSs, further optimisations to reduce the response times to detected attacks and eliminate software overheads on the ECU through efficient integration strategies were investigated. In the final chapter, the direct integration of a quantised IDS model is discussed within the datapath of an otherwise standard CAN network controller.

The datapath on the receive side of the controller was extended to embed a quantised IDS into the datapath, allowing the byte-decoded data from the CAN bus to be fed directly into the IDS. This allowed the IDS execution to overlap with the protocol checks, thus enabling early attack detection and flagging of malicious messages before they were transferred to the receiving ECU. This approach allows ECUs to enter safe operating modes immediately, reducing potential damage from cyberattacks, and eliminating any software changes/overheads for IDS functionality.

7.7 Concluding remarks

IDS systems are envisioned as the first line of defence for in-vehicle systems, where multiple network standards and protocol systems coexist to achieve safety-critical and autonomous functions. The proposed strategies in this thesis enhance the state of IDS solutions proposed for in-vehicle networks by jointly addressing performance requirements (high detection accuracy, low false positives) and deployment constraints (energy and resource consumption, software overheads, detection latency) for the range of CAN deployment conditions (low-speed/high-speed).

The thesis proposed the development of quantised neural networks as the IDS solutions in chapter 3, with significant gains in energy efficiency and resource consumption compared to the state-of-the-art, while bettering the detection accuracy across different attack vectors. Chapter 4 proposed a generalised anomaly detection model that learns normal message sequences of a CAN system to efficiently identify threat vectors that cause deviations to normal traffic flow on the bus and demonstrates its efficiency across multiple datasets. The quantised autoencoder model also exploits model compression strategies to reduce resource, energy and latency overheads. A novel quantised LSTM-based model was developed in chapter 5 which eliminates the requirements of block-based inference for existing schemes. Coupled with an intelligent integration, the model offers the best balance between performance requirements and constraints for real-world solutions. Finally, in chapter 6, a novel integration strategy for network-offloaded IDS is evaluated using a quantised MLP model as the IDS engine. The developed SecCAN controller effectively implements IDS at the interface, making it fully transparent to the ECU while achieving line rate detection at the highest data rates. The next section discusses some future research directions that stem from this thesis.

7.8 Future Research

The contributions of this thesis open up several promising avenues for future research in automotive cybersecurity, particularly in the design and integration of advanced IDS/IPS solutions within vehicular networks and embedded ECU architectures.

7.8.1 Real-Time Intrusion Prevention and Mitigation Strategies

A key direction for future work is the development of Intrusion Prevention Systems (IPS) that not only detect threats but also respond to them in real-time. This involves designing:

- Response mechanisms that allow vehicles to transition into predefined safe operating modes upon detecting intrusions.
- Isolation techniques to minimise the impact of compromised ECUs within the CAN network, improving overall system resilience.

Given that IPSs often exploit protocol-level semantics to detect and block malicious traffic, future research could investigate implementing IPS logic on the ARM cores within the PS of the ZCU104 SoC. This would enable a multi-stage intrusion detection and prevention pipeline on a single SoC, reducing hardware complexity and improving latency. Such an architecture is well-suited for automotive applications, where power, space, and computational resources are severely constrained.

7.8.2 Integrated and Holistic Cybersecurity Frameworks

Another promising direction is the integration of IDSs with proactive prevention mechanisms to enable a more holistic cybersecurity framework. As vehicles become increasingly autonomous and interconnected, integrated IDS/IPS systems will be essential for ensuring dependable protection against sophisticated and evolving cyber threats. This approach is particularly important in the context of ADAS and automated driving platforms, where both safety and reliability are critical.

7.8.3 Dataset Expansion and Model Generalisation

One of the critical limitations in current IDS/IPS research is the lack of comprehensive, publicly available attack datasets. Existing datasets often cover only a narrow range of attack types and scenarios, which restricts the generalisability of machine learning-based models. Future research should prioritise:

- Building richer, more diverse CAN attack datasets that span a broader range of threat scenarios, vehicle types, and operational environments.
- Enabling comprehensive testing and benchmarking of IDS/IPS models under realistic and varied conditions.

Such datasets would not only strengthen the scientific validity of existing models but also help establish their real world viability and robustness. This is especially critical as the automotive industry transitions towards AI-integrated platforms for autonomous driving, predictive maintenance, and infotainment systems [165].

7.8.4 Federated Learning for Distributed Threat Detection

Another future direction is the adoption of Federated Learning (FL) for decentralised, privacy-preserving threat detection. In this approach:

- Vehicles collaboratively train global IDS models while keeping sensitive data local.
- Fleet-level insights can be shared without compromising proprietary or personal information.

Such distributed learning architectures enhance the adaptability of IDSs across a fleet and enable faster learning from emerging threats.

7.8.5 Threat Intelligence Propagation and Knowledge Sharing

To further improve the responsiveness of automotive cybersecurity systems, future work can explore:

- Efficient mechanisms for threat knowledge dissemination across vehicle fleets.
- Leveraging OTA updates and edge computing to propagate security updates in real-time.

These capabilities will be essential for enabling rapid response to new threats and enhancing the overall cyber-resilience of connected vehicle ecosystems.

7.8.6 Adaptation to Emerging E/E Architectures

Finally, future IDS/IPS designs must consider the evolution of vehicle networks and E/E architectures, including:

- **Service-Oriented Architectures (SOA)** and **Zonal Architectures**, which decentralise and virtualise traditional ECU functionalities.

These emerging paradigms introduce new challenges and opportunities for distributing security functions and integrating IDS/IPS mechanisms in more flexible and scalable ways.

7.8.7 Concluding Remarks

In this thesis, a suite of IDS models tailored for automotive CAN networks has been proposed, designed, and rigorously evaluated. The work addresses deployment challenges such as low latency, energy efficiency, and resource constraints, critical for real-time embedded systems in vehicles. Deployment strategies, including optimal

IDS placement within the CAN ecosystem, have also been investigated to balance detection accuracy and system overhead.

Overall, this thesis contributes practical, deployable IDS solutions that bridge the gap between academic research and real-world automotive requirements. The models developed herein strike a balance between security robustness and operational efficiency, laying the groundwork for safer, more resilient vehicular networks.

References

- [1] Mourad Limam, Thomas Eymann, and Bernard Bäker. Automotive e/e network architectures: Evolution, trends and future challenges. *Moderne Elektronik im Kraftfahrzeug*, 2:23, 2007.
- [2] Kamil Çağatay Bayindir, Mehmet Ali Gözükcük, and Ahmet Teke. A comprehensive overview of hybrid electric vehicle: Powertrain configurations, powertrain control techniques and electronic control units. *Energy conversion and Management*, 52(2):1305–1313, 2011.
- [3] OF Vynakov, EV Savolova, and AI Skrynnyk. Modern electric cars of tesla motors company. *Automation of technological and business processes*, 8(2), 2016.
- [4] Xin Li and Kyung-Min Nam. Environmental regulations as industrial policy: Vehicle emission standards and automotive industry performance. *Environmental Science & Policy*, 131:68–83, 2022.
- [5] Jaswanth Nidamanuri, Chinmayi Nibhanupudi, Rolf Assfalg, and Hrishikesh Venkataraman. A progressive review: Emerging technologies for ADAS driven solutions. *IEEE Transactions on Intelligent Vehicles*, 7(2):326–341, 2021.
- [6] Bernhard Pauli and Arno Meyna. Reliability of electronic control units in motor vehicles. *SAE transactions*, pages 630–637, 1998.
- [7] R. B. GmbH. CAN Specification, Version 2.0, 1991.
- [8] Andrzej Sumorek and Marcin Buczaj. The evolution of “media oriented systems transport” protocol. *Teka Komisji Motoryzacji i Energetyki Rolnictwa*, 14(3), 2014.
- [9] Matthew Ruff. Evolution of local interconnect network (LIN) solutions. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No. 03CH37484)*, volume 5, pages 3382–3389. IEEE, 2003.
- [10] Marco Di Natale, Haibo Zeng, Paolo Giusto, and Arkadeb Ghosal. *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media, 2012.

- [11] Amrita Ghosal and Mauro Conti. Security issues and challenges in V2X: A survey. *Computer Networks*, 169:107093, 2020.
- [12] Swaroop Darbha, Shyamprasad Konduri, and Prabhakar R Pagilla. Benefits of V2V communication for autonomous and connected vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 20(5):1954–1963, 2018.
- [13] Lingyun Xiao and Feng Gao. A comprehensive review of the development of adaptive cruise control systems. *Vehicle system dynamics*, 48(10):1167–1192, 2010.
- [14] Peter Hank, Steffen Müller, Ovidiu Vermesan, and Jeroen Van Den Keybus. Automotive ethernet: in-vehicle networking and smart mobility. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1735–1739. IEEE, 2013.
- [15] Tae Un Kang, Hyun Min Song, Seonghoon Jeong, and Huy Kang Kim. Automated reverse engineering and attack for CAN using OBD-II. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–7. IEEE, 2018.
- [16] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. *Def Con*, 21(260-264):15–31, 2013.
- [17] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91), 2015.
- [18] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.
- [19] Ning Lu, Nan Cheng, Ning Zhang, Xuemin Shen, and Jon W Mark. Connected vehicles: Solutions and challenges. *IEEE internet of things journal*, 1(4): 289–299, 2014.
- [20] Shashwat Khandelwal, Eashan Wadhwa, and Shanker Shreejith. Deep learning-based embedded intrusion detection system for automotive CAN. In *2022 IEEE 33rd international conference on application-specific systems, architectures and processors (ASAP)*, pages 88–92. IEEE, 2022.
- [21] Gianpiero Costantino, Marco De Vincenzi, and Ilaria Matteucci. A comparative analysis of unece wp. 29 r155 and ISO/SAE 21434. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 340–347. IEEE, 2022.
- [22] Christopher A Kahn. National highway traffic safety administration (NHTSA) notes. *Ann. Emerg. Med.*, 66:669–669, 2015.

- [23] Li Yang, Abdallah Moubayed, and Abdallah Shami. MTH-IDS: A Multitiered Hybrid Intrusion Detection System for Internet of Vehicles. *IEEE Internet of Things Journal*, 9(1):616–632, 2021.
- [24] Siti-Farhana Lokman, Abu Talib Othman, and Muhammad-Husaini Abu-Bakar. Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–17, 2019.
- [25] Francisco Eiras, Aleksandar Petrov, Bertie Vidgen, Christian Schroeder, Fabio Pizzati, Katherine Elkins, Supratik Mukhopadhyay, Adel Bibi, Aaron Purewal, Csaba Botos, et al. Risks and opportunities of open-source generative ai. *arXiv preprint arXiv:2405.08597*, 2024.
- [26] Vikash Kumar and Ditipriya Sinha. A robust intelligent zero-day cyber-attack detection technique. *Complex & Intelligent Systems*, 7(5):2211–2234, 2021.
- [27] Jan Lauinger, Mohammad Hamad, and Sebastian Steinhorst. Toward a Multi-Layer Intrusion Response System for Connected Vehicles.
- [28] Subir Halder, Mauro Conti, and Sajal K Das. COIDS: A clock offset based intrusion detection system for controller area networks. In *Proceedings of the 21st International Conference on Distributed Computing and Networking*, pages 1–10, 2020.
- [29] Mehmet Bozdal, Mohammad Samie, and Ian K Jennions. WINDS: A wavelet-based intrusion detection system for controller area network (CAN). *IEEE Access*, 9:58621–58633, 2021.
- [30] Sampath Rajapaksha, Harsha Kalutarage, M Omar Al-Kadri, Andrei Petrovski, Garikayi Madzudzo, and Madeline Cheah. Ai-based intrusion detection systems for in-vehicle networks: A survey. *ACM Computing Surveys*, 55(11):1–40, 2023.
- [31] Araya Kibrom Desta, Shuji Ohira, Ismail Arai, and Kazutoshi Fujikawa. MLIDS: Handling Raw High-Dimensional CAN Bus Data Using Long Short-Term Memory Networks for Intrusion Detection in In-Vehicle Networks. In *Proc. Intl. Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–7. IEEE, 2020.
- [32] Araya Kibrom Desta, Shuji Ohira, Ismail Arai, and Kazutoshi Fujikawa. Rec-CNN: In-vehicle networks intrusion detection using convolutional neural networks trained on recurrence plots. *Vehicular Communications*, 35:100470, 2022.
- [33] Shanker Shreejith, Bezborah Anshuman, and Suhaib A Fahmy. Accelerated artificial neural networks on FPGA for fault detection in automotive systems. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 37–42. IEEE, 2016.

- [34] Shanker Shreejith, Suhaib A Fahmy, and Martin Lukasiewicz. Reconfigurable computing in next-generation automotive networks. *IEEE Embedded Systems Letters*, 5(1):12–15, 2013.
- [35] Electronic Control Unit in Automotive Systems. <https://www.totalphase.com/blog/2024/05/what-is-electronic-control-unit-automotive-systems/>, 2024.
- [36] Georg Gut, Christian Allmann, Markus Schurius, and Karsten Schmidt. Reduction of electronic control units in electric vehicles using multicore technology. In *Multicore Software Engineering, Performance, and Tools: International Conference, MSEPT 2012, Prague, Czech Republic, May 31-June 1, 2012. Proceedings 1*, pages 90–93. Springer, 2012.
- [37] Florian Hartwich et al. CAN with flexible data-rate. In *Proc. iCC*, pages 1–9. Citeseer, 2012.
- [38] Karl Henrik Johansson, Martin Törngren, and Lars Nielsen. Vehicle applications of controller area network. *Handbook of networked and embedded control systems*, pages 741–765, 2005.
- [39] Masaru Takada, Yuki Osada, and Masakatu Morii. Counter attack against the bus-off attack on can. In *2019 14th Asia Joint Conference on Information Security (AsiaJCIS)*, pages 96–102. IEEE, 2019.
- [40] Gianluca Cena, Stefano Scanzio, and Adriano Valenzano. Composite CAN XL-Ethernet Networks for Next-Gen Automotive and Automation Systems. In *2023 IEEE 19th International Conference on Factory Communication Systems (WFCS)*, pages 1–8. IEEE, 2023.
- [41] Mehmet Bozdal, Mohammad Samie, and Ian Jennions. A Survey on CAN Bus Protocol: Attacks, Challenges, and Potential Solutions. In *Proc. Intl. Conf. on Computing, Electronics Communications Engineering (iCCECE)*, pages 201–205, 2018. doi: 10.1109/iCCECOME.2018.8658720.
- [42] Subhojeet Mukherjee, Hossein Shirazi, Indrakshi Ray, Jeremy Daily, and Rose Gamble. Practical DoS attacks on embedded networks in commercial vehicles. In *Proc. Intl Conference on Information Systems Security*, pages 23–42. Springer, 2016.
- [43] Miro Enev, Alex Takakuwa, Karl Koscher, and Tadayoshi Kohno. Automobile Driver Fingerprinting. *Proc. Priv. Enhancing Technol.*, 2016(1):34–50, 2016.
- [44] Karl Koscher, Stefan Savage, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Alexei Czeskis, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Proc. IEEE Symposium on Security and Privacy*, pages 447–462. IEEE Computer Society, 2010.

- [45] Andrea Palanca, Eric Evenchick, Federico Maggi, and Stefano Zanero. A stealth, selective, link-layer denial-of-service attack against automotive networks. In *Proc. Intl. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 185–206. Springer, 2017.
- [46] Xinpeng Hao and Xiangxue Li. FirmCAN: Sensitive CAN Knowledge Leakage from Automotive ECUs. *ACM Transactions on Embedded Computing Systems*, 2025.
- [47] Zhaogang Qi. Advances on air conditioning and heat pump system in electric vehicles—A review. *Renewable and Sustainable Energy Reviews*, 38:754–764, 2014.
- [48] Charles M Kozierok, Robert B Boatright, Jeffrey Quesnelle, and Colt Correa. Automotive Ethernet-The Definitive Guide. 2014.
- [49] Keeping in pace with in-vehicle data. <https://www.sae.org/news/2022/01/keeping-pace-with-in-vehicle-data-speed>, 2024.
- [50] Aida Ben Chehida Douss, Ryma Abassi, and Damien Sauveron. State-of-the-art survey of in-vehicle protocols and automotive Ethernet security and vulnerabilities. *Mathematical Biosciences and Engineering*, 20(9):17057–17095, 2023.
- [51] Automotive Ethernet: The In-Vehicle Networking of the Future. <https://www.keysight.com/blogs/en/tech/educ/2024/automotive-ethernet>, 2024.
- [52] Andreas Kern. Ethernet and IP for automotive E/E-architectures. *Doktorarbeit, Universität Erlangen-Nürnberg*, 2012.
- [53] Rainer Makowitz and Christopher Temple. Flexray-a communication network for automotive control systems. In *2006 IEEE International Workshop on Factory Communication Systems*, pages 207–212. IEEE, 2006.
- [54] Flexray. <https://en.wikipedia.org/wiki/flexray>, 2024.
- [55] National Instruments - FlexRay Automotive Communication Bus Overview. <https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/flexray-automotive-communication-bus-overview.html>, 2024.
- [56] A Simple Introduction to LIN. <https://www.csselectronics.com/pages/lin-bus-protocol-intro-basics>, 2025.
- [57] Hans-Christian von der Wense. Introduction to local interconnect network. Technical report, SAE Technical Paper, 2000.

- [58] Microchip introduction to MOST. <https://www.microchip.com/en-us/solutions/automotive-and-transportation/automotive-products/connectivity/most-technology>, 2024.
- [59] Media Oriented Systems Transport. https://en.wikipedia.org/wiki/most_bus, 2024.
- [60] Lucia Lo Bello, Riccardo Mariani, Saad Mubeen, and Sergio Saponara. Recent advances and trends in on-board embedded and networked automotive systems. *IEEE Transactions on Industrial Informatics*, 15(2):1038–1051, 2018.
- [61] Paul Carsten, Todd R Andel, Mark Yampolskiy, and Jeffrey T McDonald. In-vehicle networks: Attacks, vulnerabilities, and proposed solutions. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, pages 1–8, 2015.
- [62] Steve Corrigan HPL. Introduction to the controller area network (CAN). *Application Report SLOA101*, pages 1–17, 2002.
- [63] Ahmed Hazem and HA Fahmy. Lcap-a lightweight can authentication protocol for securing in-vehicle networks. In *10th escar Embedded Security in Cars Conference, Berlin, Germany*, volume 6, page 172, 2012.
- [64] Ryan J Duplechin. The emerging intersection of products liability, cybersecurity, and autonomous vehicles. *Tenn. L. Rev.*, 85:803, 2017.
- [65] Gianpiero Costantino, Marco De Vincenzi, and Ilaria Matteucci. A vehicle firmware security vulnerability: an IVI exploitation. *Journal of Computer Virology and Hacking Techniques*, pages 1–16, 2024.
- [66] Sen Nie, Ling Liu, and Yuefeng Du. Free-fall: Hacking tesla from wireless to can bus. *Briefing, Black Hat USA*, 25(1):16, 2017.
- [67] Angela Gonzalez Mariño, Francesc Fons, and Juan Manuel Moreno Arostegui. The future roadmap of in-vehicle network processing: A HW-centric (R-) evolution. *IEEE access*, 10:69223–69249, 2022.
- [68] Aggelos Soteropoulos, Mathias Mitteregger, Martin Berger, and Jakob Zwirchmayr. Automated drivability: Toward an assessment of the spatial deployment of level 4 automated vehicles. *Transportation Research Part A: Policy and Practice*, 136:64–84, 2020.
- [69] Jianqiang Wang, Heye Huang, Keqiang Li, and Jun Li. Towards the unified principles for level 5 autonomous vehicles. *Engineering*, 7(9):1313–1325, 2021.
- [70] Bogdan Groza and Pal-Stefan Murvay. Security solutions for the controller area network: Bringing authentication to in-vehicle networks. *IEEE Vehicular Technology Magazine*, 13(1):40–47, 2018.

- [71] Omar Y Al-Jarrah, Carsten Maple, Mehrdad Dianati, David Oxtoby, and Alex Mouzakitis. Intrusion detection systems for intra-vehicle networks: A review. *IEEE Access*, 7:21266–21289, 2019.
- [72] Tuan Phan Vuong, George Loukas, and Diane Gan. Performance evaluation of cyber-physical intrusion detection on a robotic vehicle. In *Proc. Intl. Conf. on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 2106–2113. IEEE, 2015.
- [73] Congli Ling and Dongqin Feng. An algorithm for detection of malicious messages on CAN buses. In *Proc. Conf. on information technology and computer science. Atlantis Press*, volume 10. Citeseer, 2012.
- [74] Habeeb Olufowobi, Clinton Young, Joseph Zambreno, and Gedare Bloom. SAID-uCANT: Specification-based automotive intrusion detection using controller area network (CAN) timing. *IEEE Transactions on Vehicular Technology*, 69(2):1484–1494, 2019.
- [75] Pal-Stefan Murvay and Bogdan Groza. TIDAL-CAN: Differential timing based intrusion detection and localization for controller area network. *IEEE Access*, 8: 68895–68912, 2020.
- [76] Dario Stabili, Mirco Marchetti, and Michele Colajanni. Detecting attacks to internal vehicle networks through hamming distance. In *AEIT Intl. Annual Conference*, pages 1–6. IEEE, 2017.
- [77] Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. Using data analytics to detect anomalous states in vehicles. *arXiv preprint arXiv:1512.08048*, 2015.
- [78] Marc Weber, Simon Klug, Eric Sax, and Bastian Zimmer. Embedded hybrid anomaly detection for automotive CAN communication. In *Proc. European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.
- [79] Daksh Kumar Vasistha. *Detecting anomalies in controller area network for automobiles*. PhD thesis, 2017.
- [80] Wufei Wu, Renfa Li, Guoqi Xie, Jiyao An, Yang Bai, Jia Zhou, and Keqin Li. A Survey of Intrusion Detection for In-Vehicle Networks. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):919–933, 2020. doi: 10.1109/TITS.2019.2908074.
- [81] Bogdan Groza and Pal-Stefan Murvay. Efficient intrusion detection with bloom filtering in controller area networks. *IEEE Transactions on Information Forensics and Security*, 14(4):1037–1051, 2018.
- [82] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 911–927, 2016.

- [83] Kyong-Tak Cho and Kang G Shin. Viden: Attacker identification on in-vehicle networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1109–1123, 2017.
- [84] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 57–5709. IEEE, 2017.
- [85] Wufei Wu, Yizhi Huang, Ryo Kurachi, Gang Zeng, Guoqi Xie, Renfa Li, and Keqin Li. Sliding window optimized information entropy analysis method for intrusion detection on in-vehicle networks. *IEEE Access*, 6:45233–45245, 2018.
- [86] Aymen Boudguiga, Witold Klaudel, Antoine Boulanger, and Pascal Chiron. A simple intrusion detection method for controller area network. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2016.
- [87] Stefano Longari, Matteo Penco, Michele Carminati, and Stefano Zanero. Copycan: An error-handling protocol based intrusion detection system for controller area network. In *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy*, pages 39–50, 2019.
- [88] Habeeb Olufowobi, Sena Hounsinnou, and Gedare Bloom. Controller area network intrusion prevention system leveraging fault recovery. In *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy*, pages 63–73, 2019.
- [89] Abdulaziz Alshammari, Mohamed A Zohdy, Debatosh Debnath, and George Corser. Classification approach for intrusion detection in vehicle systems. *Wireless Engineering and Technology*, 9(4):79–94, 2018.
- [90] Li Yang, Abdallah Moubayed, Ismail Hamieh, and Abdallah Shami. Tree-based intelligent intrusion detection system in internet of vehicles. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.
- [91] Eunbi Seo, Hyun Min Song, and Huy Kang Kim. GIDS: GAN based intrusion detection system for in-vehicle network. In *Proc. Conf. on Privacy, Security and Trust (PST)*, pages 1–6. IEEE, 2018.
- [92] Pengzhou Cheng, Kai Xu, Simin Li, and Mu Han. TCAN-IDS: Intrusion Detection System for Internet of Vehicle Using Temporal Convolutional Attention Network. *Symmetry*, 14(2):310, 2022.
- [93] Kushagra Agrawal, Tejasvi Alladi, Ayush Agrawal, Vinay Chamola, and Abderrahim Benslimane. NovelADS: A Novel Anomaly Detection System for Intra-Vehicular Networks. *IEEE Transactions on Intelligent Transportation Systems*, 2022.
- [94] Haoyu Ma, Jianqiu Cao, Bo Mi, Darong Huang, Yang Liu, and Shaoqian Li. A GRU-Based Lightweight System for CAN Intrusion Detection in Real Time. *Security and Communication Networks*, 2022, 2022.

- [95] Paulo Freitas De Araujo-Filho, Antônio J Pinheiro, Georges Kaddoum, Divanilson R Campelo, and Fabio L Soares. An Efficient Intrusion Prevention System for CAN: Hindering Cyber-Attacks with a Low-Cost Platform. *IEEE Access*, 9:166855–166869, 2021.
- [96] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. Anomaly intrusion detection method for vehicular networks based on survival analysis. *Vehicular communications*, 14:52–63, 2018.
- [97] everythingPE. <https://www.everythingpe.com/community/what-is-power-rating-of-an-ev-ecu>, 2024.
- [98] Electronic Design. <https://www.electronicdesign.com/technologies/embedded/digital-ics/processors/microcontrollers/article/21798774/high-performance-32-bit-microcontroller-delivers-330-dmips>, 2013.
- [99] Aspencore Network. <https://www.edn.com/25000-dmips-capable-soc-targets-the-automotive-infotainment-market/>, 2013.
- [100] JJ Wang, RB Katz, JS Sun, BE Cronquist, JL McCollum, TM Speers, and WC Plants. SRAM based re-programmable FPGA for space applications. *IEEE Transactions on Nuclear Science*, 46(6):1728–1735, 1999.
- [101] Michael Wirthlin. High-reliability FPGA-based systems: Space, high-energy physics, and beyond. *Proceedings of the IEEE*, 103(3):379–389, 2015.
- [102] Juan J Rodríguez-Andina, Maria D Valdes-Pena, and Maria J Moure. Advanced features and industrial applications of FPGAs—A review. *IEEE Transactions on Industrial Informatics*, 11(4):853–864, 2015.
- [103] N Pavan Kumar and VH Patankar. Battery-powered FPGA-based embedded system for ultrasonic pipe inspection and gauging systems. *Review of Scientific Instruments*, 94(3), 2023.
- [104] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan, and Yu-Wing Tai. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In *Proc. Design Automation Conference*, pages 1–6, 2017.
- [105] Teng Wang, Chao Wang, Xuehai Zhou, and Huaping Chen. A survey of fpga based deep learning accelerators: Challenges and opportunities. *arXiv preprint arXiv:1901.04988*, pages 1–10, 2018.
- [106] A survey and taxonomy of FPGA-based deep learning accelerators. *Journal of Systems Architecture*, 98:331–345, 2019.
- [107] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, 2016.

- [108] Hardik Sharma, Jongse Park, Emmanuel Amaro, Bradley Thwaites, Praneetha Kotha, Anmol Gupta, Joon Kyung Kim, Asit Mishra, and Hadi Esmaeilzadeh. Dnnweaver: From high-level deep network models to FPGA acceleration. In *The Workshop on Cognitive Architectures*, 2016.
- [109] Erwei Wang, James J Davis, Peter YK Cheung, and George A Constantinides. LUTNet: Rethinking inference in FPGA soft logic. In *Proc. Intl. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 26–34. IEEE, 2019.
- [110] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proc. Intl. Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 65–74, 2017.
- [111] Xilinx. Vitis AI User Guide, 2021.
- [112] Xilinx. Zynq DPU v3.2, 2020.
- [113] Shreejith, Shanker and Fahmy, Suhaib A. and Lukasiewicz, Martin. Reconfigurable Computing in Next-Generation Automotive Networks. *IEEE Embedded Systems Letters*, 5(1):12–15, 2013. doi: 10.1109/LES.2013.2243698.
- [114] Shanker Shreejith and Suhaib A. Fahmy. Extensible FlexRay Communication Controller for FPGA-Based Automotive Systems. *IEEE Transactions on Vehicular Technology*, 64(2):453–465, 2015. doi: 10.1109/TVT.2014.2324532.
- [115] Shanker Shreejith, Philipp Mundhenk, Andreas Ettner, Suhaib A Fahmy, Sebastian Steinhorst, Martin Lukasiewicz, and Samarjit Chakraborty. VEGa: A high performance vehicular ethernet gateway on hybrid FPGA. *IEEE Transactions on Computers*, 66(10):1790–1803, 2017.
- [116] Shanker Shreejith, Kizhepatt Vipin, Suhaib A Fahmy, and Martin Lukasiewicz. An approach for redundancy in FlexRay networks using FPGA partial reconfiguration. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 721–724. IEEE, 2013.
- [117] Shanker Shreejith and Suhaib A Fahmy. Smart network interfaces for advanced automotive applications. *IEEE Micro*, 38(2):72–80, 2018.
- [118] Shanker Shreejith and Suhaib A Fahmy. Zero latency encryption with FPGAs for secure time-triggered automotive networks. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 256–259. IEEE, 2014.
- [119] Francisco Fons and Mariano Fons. FPGA-based automotive ECU design addresses AUTOSAR and ISO 26262 standards. *Xcell journal*, 78:20, 2012.
- [120] Robert Bosch GmbH. *Engine Control Unit MS 6*. Robert Bosch GmbH,, 2015.

- [121] Porsche Engineering. <https://www.porscheengineering.com/en/peg/newsroom/pem-2-24/turning-dc-into-ac/>, 2025.
- [122] Jia Zhou, Prachi Joshi, Haibo Zeng, and Renfa Li. Btmonitor: Bit-time-based intrusion detection and attacker identification in controller area network. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(6):1–23, 2019.
- [123] Yun Yang, Zongtao Duan, and Mark Tehranipoor. Identify a Spoofing Attack on an In-Vehicle CAN Bus Based on the Deep Features of an ECU Fingerprint Signal. *Smart Cities*, 3(1):17–30, 2020.
- [124] Mario Casillo, Simone Coppola, Massimo De Santo, Francesco Pascale, and Emanuele Santonicola. Embedded Intrusion Detection System for Detecting Attacks over CAN-BUS. In *Proc. Intl. Conf. on System Reliability and Safety (ICSRS)*, pages 136–141, 2019. doi: 10.1109/ICSRS48664.2019.8987605.
- [125] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.
- [126] Xilinx. CAN FD 3.0 User Guide, 2021.
- [127] Roger Luis, L Enrique Sucar, and Eduardo F Morales. Inductive transfer for learning Bayesian networks. *Machine learning*, 79:227–255, 2010.
- [128] CAR Hacking Dataset. <https://ocslab.hksecurity.net/datasets/can-intrusion-dataset>, 2020.
- [129] Alessandro Pappalardo. Xilinx/brevitas, 2021. URL <https://doi.org/10.5281/zenodo.3333552>.
- [130] Wei Lo, Hamed Alqahtani, Kutub Thakur, Ahmad Almadhor, Subhash Chander, and Gulshan Kumar. A hybrid deep learning based intrusion detection system using spatial-temporal representation of in-vehicle network traffic. *Vehicular Communications*, 35:100471, 2022.
- [131] Sung Bum Park, Hyo Jin Jo, and Dong Hoon Lee. G-IDCS: Graph-Based Intrusion Detection and Classification System for CAN Protocol. *IEEE Access*, 11:39213–39227, 2023. doi: 10.1109/ACCESS.2023.3268519.
- [132] Trieu Phong Nguyen, Heungwoo Nam, and Daehee Kim. Transformer-Based Attention Network for In-Vehicle Intrusion Detection. *IEEE Access*, 11:55389–55403, 2023. doi: 10.1109/ACCESS.2023.3282110.
- [133] Shashwat Khandelwal and Shanker Shreejith. A lightweight FPGA-based IDS-ECU architecture for automotive CAN. In *2022 international conference on field-programmable technology (ICFPT)*, pages 1–9. IEEE, 2022.
- [134] Upstream Report. <https://upstream.auto/reports/global-automotive-cybersecurity-report/>, 2023.

- [135] Harini Narasimhan, Vinayakumar Ravi, and Nazeeruddin Mohammad. Unsupervised deep learning approach for in-vehicle intrusion detection system. *IEEE Consumer Electronics Magazine*, 12(1):103–108, 2021.
- [136] Afia Anjum, Paul Agbaje, Sena Hounsinnou, and Habeeb Olufowobi. In-vehicle network anomaly detection using extreme gradient boosting machine. In *Proceedings of the Mediterranean Conference on Embedded Computing (MECO)*, pages 1–6. IEEE, 2022.
- [137] Md Delwar Hossain, Hiroyuki Inoue, Hideya Ochiai, Doudou Fall, and Youki Kadobayashi. LSTM-based intrusion detection system for in-vehicle can bus communications. *IEEE Access*, 8:185489–185502, 2020.
- [138] Vita Santa Barletta, Danilo Caivano, Mirko De Vincentiis, and Antonio Piccinno. Black Mamba: A Multi-class Random Forest for Automotive Intrusion Detection.
- [139] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proc. of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, 2014.
- [140] Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th international conference on data mining workshops (ICDMW)*, pages 241–246. IEEE, 2016.
- [141] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. *Advances in neural information processing systems*, 26, 2013.
- [142] Ekaterina Govorkova, Ema Puljak, Thea Aarrestad, Thomas James, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, Nicolò Ghielmetti, Maksymilian Graczyk, Sioni Summers, et al. Autoencoders on field-programmable gate arrays for real-time, unsupervised new physics detection at 40 mhz at the large hadron collider. *Nature Machine Intelligence*, 4(2):154–161, 2022.
- [143] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [144] ONNX. <https://onnx.ai/>.
- [145] ONNX. <https://github.com/onnx/onnx/blob/main/docs/operators.md#scan>, 2018.
- [146] Alessandro Pappalardo, Yaman Umuroglu, Michaela Blott, Jovan Mitrevski, Ben Hawks, Nhan Tran, Vladimir Loncar, Sioni Summers, Hendrik Borrás, Jules Muhizi, et al. Qonnx: Representing arbitrary-precision quantized neural networks. *arXiv preprint arXiv:2206.07527*, 2022.
- [147] Tobias Alonso, Lucian Petrica, Mario Ruiz, Jakoba Petri-Koenig, Yaman Umuroglu, Ioannis Stamelos, Elias Koromilas, Michaela Blott, and Kees Vissers. Elastic-df:

- Scaling performance of dnn inference in fpga clouds through automatic partitioning. *ACM Trans. Reconfigurable Technol. Syst.*, 15(2), December 2021. ISSN 1936-7406. doi: 10.1145/3470567. URL <https://doi.org/10.1145/3470567>.
- [148] Vladimir Rybalkin, Alessandro Pappalardo, Muhammad Mohsin Ghaffar, Giulio Gambardella, Norbert Wehn, and Michaela Blott. FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs. In *2018 28th international conference on field programmable logic and applications (FPL)*, pages 89–897. IEEE, 2018.
- [149] Auangkun Rangsikunpum, Sam Amiri, and Luciano Ost. BIDS: An efficient Intrusion Detection System for in-vehicle networks using a two-stage Binarised Neural Network on low-cost FPGA. *Journal of Systems Architecture*, 156:103285, 2024.
- [150] Shashwat Khandelwal and Shanker Shreejith. Exploring Highly Quantised Neural Networks for Intrusion Detection in Automotive CAN. In *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, pages 235–241. IEEE, 2023.
- [151] Amit Chougule, Ishan Kulkarni, Tejasvi Alladi, Vinay Chamola, and Fei Richard Yu. HybridSecNet: In-Vehicle Security on Controller Area Networks Through a Hybrid Two-Step LSTM-CNN Model. *IEEE Transactions on Vehicular Technology*, 73(10): 14580–14591, 2024. doi: 10.1109/TVT.2024.3413849.
- [152] Kavya Balasubramanian, Adithya Gowda Baragur, Denis Donadel, Dinuka Saha-bandu, Alessandro Brighente, Bhaskar Ramasubramanian, Mauro Conti, and Radha Poovendran. CANLP: NLP-Based Intrusion Detection System for CAN. In *Proceedings of the ACM/SIGAPP Symposium on Applied Computing*, pages 212–214, 2024.
- [153] Linxi Zhang, Xuke Yan, and Di Ma. A Binarized Neural Network Approach to Accelerate in-Vehicle Network Intrusion Detection. *IEEE Access*, 10:123505–123520, 2022. doi: 10.1109/ACCESS.2022.3208091.
- [154] Paulo Freitas De Araujo-Filho, Antônio J. Pinheiro, Georges Kaddoum, Divanilson R. Campelo, and Fabio L. Soares. An Efficient Intrusion Prevention System for CAN: Hindering Cyber-Attacks With a Low-Cost Platform. *IEEE Access*, 9:166855–166869, 2021. doi: 10.1109/ACCESS.2021.3136147.
- [155] Hongqian Wei, Qiang Ai, Wenqiang Zhao, and Youtong Zhang. Real-time security warning and ECU identification for in-vehicle networks. *IEEE Sensors Journal*, 23(17):20258–20266, 2023.
- [156] Azeem Hafeez, Janani Mohan, Mansi Girdhar, and Selim S Awad. Machine learning based ecu detection for automotive security. In *2021 17th International Computer Engineering Conference (ICENCO)*, pages 73–81. IEEE, 2021.

-
- [157] Tudor Andreica, Christian-Daniel Curiac, Camil Jichici, and Bogdan Groza. Android head units vs. In-vehicle ECUs: performance assessment for deploying in-vehicle intrusion detection systems for the CAN bus. *IEEE Access*, 10:95161–95178, 2022.
- [158] Marco Bresch and Noräs Salman. Design and implementation of an intrusion detection system (IDS) for in-vehicle networks. 2017.
- [159] Armin Wasicek, Mert D Pesé, André Weimerskirch, Yelizaveta Burakova, and Karan Singh. Context-aware intrusion detection in automotive control systems. In *Proc. 5th ESCAR USA Conf*, pages 21–22, 2017.
- [160] Shashwat Khandelwal, Anneliese Walsh, and Shanker Shreejith. Quantised Neural Network Accelerators for Low-Power IDS in Automotive Networks. In *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–2, 2023. doi: 10.23919/DATE56975.2023.10137016.
- [161] Opencorecan. <https://opencores.org/projects/can>, 2017.
- [162] Afia Anjum, Paul Agbaje, Sena Hounsinnou, and Habeeb Olufowobi. In-Vehicle Network Anomaly Detection Using Extreme Gradient Boosting Machine. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–6, 2022. doi: 10.1109/MECO55406.2022.9797224.
- [163] Md Delwar Hossain, Hiroyuki Inoue, Hideya Ochiai, Doudou Fall, and Youki Kadobayashi. LSTM-Based Intrusion Detection System for In-Vehicle Can Bus Communications. *IEEE Access*, 8:185489–185502, 2020. doi: 10.1109/ACCESS.2020.3029307.
- [164] Shashwat Khandelwal and Shanker Shreejith. Real-time zero-day intrusion detection system for automotive controller area network on FPGAs. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 139–146. IEEE, 2023.
- [165] Automotive Compute Subsystems, ARM Zena CSS. <https://www.arm.com/products/automotive/compute-subsystems/zena>, 2025.