A Lightweight FPGA-based IDS-ECU Architecture for Automotive CAN

Shashwat Khandelwal & Shanker Shreejith
Department of Electronic & Electrical Engineering, Trinity College Dublin
Dublin, Ireland
Email: {khandels, shankers}@tcd.ie

2022 International Conference on Field-Programmable Technology (ICFPT) | 978-1-6654-5336-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICFPT56656.2022.9974508

***Abstract*—Recent years have seen an exponential rise in complex software-driven functionality in vehicles, leading to a rising number of electronic control units (ECUs), network capabilities, and interfaces. These expanded capabilities also bring-in new planes of vulnerabilities making intrusion detection and management a critical capability; however, this can often result in more ECUs and network elements due to the high computational overheads. In this paper, we present a consolidated ECU architecture incorporating an Intrusion Detection System (IDS) for Automotive Controller Area Network (CAN) along with traditional ECU functionality on an off-the-shelf hybrid FPGA device, with near-zero overhead for the ECU functionality. We propose two quantised multi-layer perceptrons (QMLP's) as isolated IDSs for detecting a range of attack vectors including Denial-of-Service, Fuzzing and Spoofing, which are accelerated using off-the-shelf deep-learning processing unit (DPU) IP block from Xilinx, operating fully transparently to the software on the ECU. The proposed models achieve the state-of-the-art classification accuracy for all the attacks, while we observed a 15× reduction in power consumption when compared against the GPU-based implementation of the same models quantised using Nvidia libraries. We also achieved a 2.3× speed up in per-message processing latency (at 0.24 ms from the arrival of a CAN message) to meet the strict end-to-end latency on critical CAN nodes and a 2.6× reduction in power consumption for inference when compared to the state-of-the-art IDS models on embedded IDS and loosely coupled IDS accelerators (GPUs) discussed in the literature.**

***Index Terms*—Controller Area Network, Intrusion Detection System, Machine Learning, Field Programmable Gate Arrays**

I. Introduction

Automotive networks are continually increasing in size and complexity as advanced electronic systems like driver assistance systems (ADAS) and high-bandwidth sensors are becoming increasingly common on most production vehicles. Furthermore, such systems are offering increased connectivity to the outside world to enable new waves of safety and comfort applications. Such increased connectivity of various electronic control units (ECUs) (and the vehicular networks these communicate over) allows remote monitoring and control of critical systems for diagnostics, software services and over-the-air upgrades. However, while they enable constant upgrade and upkeep of vehicles to improve their longevity, such interfaces also opened new avenues for attackers to deploy both invasive and non-invasive schemes to inject malicious content on these previously siloed internal networks [1]–[3]. Attacks that involve taking control of the vehicular network leading to control its critical functions have been demonstrated by multiple research groups [4]. These attacks have been made possible because of the inherent lack of security features in legacy networks like Controller Area Network (CAN) [5] that are used in vehicular networks for critical information exchange among ECUs. With wireless networks and vehicle-to-X communication becoming increasingly common, more pathways emerge for attackers to launch multi-vehicle attacks without requiring physical access or tampered ECUs.

Multiple mitigation strategies have been proposed to address these challenges, prominent among which are intrusion detection systems (IDSs). These systems integrate as an ECU on the network and analyse the flow of network traffic including the message contents in vehicle networks looking for unusual network activities or discrepancies. Any unusual behaviour is flagged as a possible threat, the severity of which is evaluated by critical systems to determine appropriate actions ranging from alerting the user to falling back to a minimal 'safe operating mode'. Early IDSs relied on rules/specifications [6], [7] that captured parameters of network and messages under standard operating mode or detected signatures of previously known attacks based [8] to determine unusual activities. However, these IDSs suffered from multiple issues including higher false positive and false negative detection, inability to scale to newer attack and memory overheads for each new rule or signature. Recently, machine learning (ML) based IDSs have shown as a viable alternative with their higher detection accuracy, adaptability to newer attack modes and their generalisable nature [9]–[14]. Despite the performance, their deployment in electric/electronic (E/E) systems continues to be non-trivial due to a combination of complex multi-standard network architecture (automotive Ethernet, CAN, FlexRay and other network protocols), complex ML models for IDS with provisions for over-the-air updates, tight power budget restricting the use of powerful GPUs for near-line-rate detection of threats and the weight/cabling overheads due to the preferred integration architecture of standalone IDS ECUs. The lack of functional consolidation stems from the inability to achieve clean resource partition between (critical) tasks on automotive (multicore) processor-based ECUs; however, loosely coupled accelerators (like GPUs) have been explored to cater to the complexity of adaptive systems like ADAS although these incur much higher power consumption and interfacing complexities.

Alternatively, ECU architectures based on hybrid FPGAs have shown the ability to achieve clear isolation between consolidated tasks on the same die, while also allowing specialised accelerators to be closely coupled to the functions improving their performance and energy efficiency. On a hybrid FPGA, a specialised hardware-efficient accelerator can accelerate the ML-IDS in isolation, while the capable ARM cores (with any custom accelerator) performs the ECU function on the same die, allowing seamless integration of distributed IDS capabilities. Prior research has explored the case for (hybrid) FPGA-based ECUs to enable compute acceleration of complex tasks in vehicular systems [15]–[17], functional consolidation [18] and reliability [19].

In this paper, we define an ECU architecture for enabling distributed IDS in vehicular networks with the IDS function deployed in isolation from the main function while retaining software control of its execution. Further, we explore a lightweight feed-forward ML model for IDS and deploy them through off-the-shelf deep-learning accelerator (Xilinx DPU) on a Zynq Ultrascale+ hybrid FPGA platform. The key contributions are as follows:

- We present a custom 8-bit feed-forward quantised multi-layer-perceptron (QMLP) based-IDS for automotive CAN achieving state-of-the-art classification accuracy across multiple attack vectors, all of which are detected using a single IDS ECU.
- The IDS-integrated ECU architecture where the IDS is deployed using off-the-shelf Xilinx DPUs modelling an AUTOSAR compliant architecture while allowing fully isolated execution of IDS task.
- The tightly integrated approach achieves noted improvements in terms of per-message processing latency and power consumption against the state-of-the-art IDSs in literature and when the quantised version of the model is implemented using loosely coupled accelerators like a GPU.

We evaluate our model and integration efficiency using the open CAR Hacking dataset with the entire CAN data frame used as an input feature to improve the detection performance. Our experiments show that the proposed lightweight QMLP-IDS achieves an average accuracy of 99.96% across multiple attack vectors such as Denial of Service (DoS), Fuzzy, and spoofing (RPM and Gear) attacks on a single device, identical to or exceeding the detection accuracy achieved by state-of-the-art GPU- and CPU-based implementations. The tightly integrated ECU architecture reduces the per message execution latency by 2.3× and the power consumed by 2.6× compared to state-of-the-art IDSs proposed in the research literature. We also see a reduction of 15× in power consumption when the quantised version of the IDS is implemented on a GPU.

The remainder of the paper is organised as follows. Section II provides background information on the CAN protocol, IDS approaches and Quantised Neural Networks; section III describes the proposed MLP model and the implemented multi-core architecture on the Ultrascale+ device; section IV outlines the experiment setup and results; and we conclude the paper in section V.

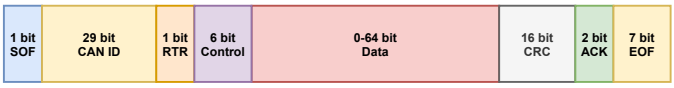

Fig. 1: Frame format of an extended frame CAN message that uses a 29-bit CAN ID.

II. Background and Related Works

A. 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 [5] and CAN-FD [20] continues to be the most widely used protocol today due to its lower cost, flexibility and robustness. Figure 1 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 a 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. The CAN ID (11-bit base, 29-bit extended format) is a unique identifier assigned to each message and by extension, defines its priority for transmission on the shared bus. The message itself is contained in the data field.

The broadcast CAN bus uses a bit-wise arbitration method to control medium access to the bus using the CAN ID allocated to each message. CAN also supports multiple data rates (125 Kbps to 1 Mpbs) and multiple modes of operation (1-wire, 2-wire) to cater to a range of critical and non-critical functions in vehicles. Despite this robustness, CAN is inherently insecure: there is no built-in mechanism in the network to authenticate the transmitter, receiver, or the message content itself [21]. This makes CAN vulnerable to simple and efficient attacks like message sniffing, fuzzing, spoofing, replay attacks, and Denial of Service (DoS) attacks [22]–[25]. These attacks rely on physical access to the CAN network; hence onset of such attacks indicate compromised ECUs on the critical network and thus detecting such intrusions are of key interest.

B. IDSs for CAN

Researchers have explored various techniques to detect intrusions in CAN bus. These can be broadly classified into flow-based, payload-based, and hybrid schemes that combine the flow-based and payload-based techniques [26]. Flow-based approaches extract identifiable traits like message frequency and/or interval for the network and use these to determine abnormal activity on the network [27]. Payload-based schemes use the information in the CAN frames to detect abnormal sequences of instructions or data [28]. Hybrid scheme uses both the timing information and the CAN frame fields to create a more holistic view, allowing them to extract specific signatures

of transmitting ECUs, receiving ECUs, and messages [29], [30]. For instance, the fingerprint-based approach uses low-level electrical signal levels and timing of signals in relation with the message contents to identify potential intrusions when large deviations are observed [31], [32]. Parameter monitoring-based techniques extract message-level frequency and timing characteristics for different fields or specific messages, like offset ratio and timing in case of a remote transmission frame, to identify potential intrusions [33]. Researchers have also proposed the use of entropy of the network traffic as a method to identify abnormal conditions and intrusions [34]. Such approaches are further generalised by machine learning based-techniques, including classification approaches, deep learning-based schemes and sequential techniques.

C. ML-based IDS

Most ML-based approaches fall under the hybrid scheme, utilizing both the content of the message frame and their timing/frequency characteristics to achieve better detection. In [35], the authors use the offset ratio and time intervals between remote requests to classify DoS and fuzzy attacks using a Support Vector Machine classifier. Tree-based approaches explored by [36] use CAN IDs and data segment contents as input features to tree-based classifiers like decision trees, random forests and others. In [10], the authors propose a reduced inception net architecture for IDS which uses deep convolutional neural networks. The authors show that the ML architecture can achieve over 99% accuracy across DoS, fuzzing, and spoofing attacks. The authors used a dataset captured from the actual vehicle for training and testing their model, which has been shared with the community for further research. Since the dataset covers multiple attack modes with actual CAN messages, we use the same dataset to train and evaluate our proposed architecture. In [9], the authors present a generative adversarial network (GAN)-based IDS trained using CAN IDs and achieve an average accuracy of 97.5% across the DoS, Fuzzy, and spoofing attacks. Our previous works have demonstrated that lightweight CNN-based IDS architectures, which uses received CAN IDs as its input feature, can achieve high classification accuracy across multiple attack vectors [37], [38]. Recently, more complex ML architectures such as temporal convolution with global attention [12], unsupervised learning through a combination of CNN and long short-term memory (LSTM) cells [11], gated recurrent units (GRU) networks [13] and CNN-based temporal dependency approaches have been shown to improve detection accuracy. These networks use CAN ID, ID + Data Field, or the entire frame as the input feature as shown in table I. In [39], the authors use an iForest 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 most cases, the inference models are deployed through powerful GPUs to achieve line rate detection on actual CAN bus (using CAN datasets), quantifying detection accuracy and processing latency (in batch-style operation). Our approach aims at integrating the ML accelerators as a standard peripheral within a traditional ECU, with optimisations to achieve high accuracy and throughput, at much lower power consumption compared to traditional GPU implementations. Hence, we quantify the detection accuracy and processing latency (in batch and per-message cases), as well as an analysis on the power consumption of the integrated approach.

TABLE I: Input features used by ML-based IDSs & IPSs proposed in the research literature.

Models	Input Features Used
GIDS [9]	CAN ID
DCNN [10]	CAN ID
Rec-CNN [40]	CAN ID
iForest [39]	Data Field
MLIDS [14]	CAN ID + Data Field
NovelADS [11]	CAN ID + Data Field
TCAN-IDS [12]	CAN ID + Data Field
MTH-IDS [41]	CAN ID + Data Field
GRU [13]	CAN ID + Data Field + DLC
MLP-IDS (proposed)	CAN ID + Data Field

D. Machine Learning on FPGAs

While GPUs have been the target device of choice for most machine learning inference applications, FPGAs have shown great promise in recent years. Early efforts of deep learning accelerators on FPGAs suffered from limited on-chip memory resources, requiring most weights and biases to be stored off-chip, resulting in performance penalties compared to GPU implementations [42]. Furthermore, such accelerators had to be custom-built and optimised while also managing efficient data movement between off-chip memory and interfaces, requiring significant hardware expertise. Recent approaches for deploying deep learning inference on FPGAs have reduced this barrier, while still providing significant performance and energy benefits over GPUs [43], [44]. Tool-chains like DNN-Weaver [45], LUTNET [46] and FINN [47] can efficiently map high-level representations of ML models for executing on pre-composed hardware accelerators or into fully custom accelerators. These tools also optimise native data representation formats (float, int) through pruning and quantisation to reduce computational complexity and memory footprint required by the implementations. The loss of accuracy in such optimisations is addressed by augmenting training steps and/or post-quantisation fine-tuning. With some hardware expertise, these frameworks will allow every layer of the network to be optimised (including weights, biases, and activation functions) and right-sized (down to 1-bit in the case of FINN) with minimal impact on accuracy, generating energy-efficient high-performance implementations, which can then be connected as slave accelerators in an embedded design or used as standalone accelerators. Alternatively, vendor flows like Vitis-AI from Xilinx [48] map a pre-trained network onto a selection of pre-composed configurable deep-learning processing units (DPUs) [49] on the hardware, lowering the barrier for using

FPGA-based ML accelerators. The Vitis-AI flow quantizes and (optionally) fine-tunes the pre-trained model to generate the hardware accelerator model along with its drivers and APIs for offloading computations to it from a host code. While not fully optimised as a custom implementation generated through FINN, the Vitis-AI flow offers a compelling alternative to the GPU flow without a significant drop in energy efficiency compared to fully custom networks at this precision (int8 precision), and hence, in this paper, we utilise the Vitis-AI flow to design and deploy our IDS accelerator on the hybrid Zynq Ultrascale+ device.

FPGA-based deployments are gaining momentum in the automotive domain, with prior research having explored acceleration of real-time complex algorithms for vision-based advanced driver assistance systems (ADAS) and using network interface extensions to augment ECU functionality and security [50]. FPGA-based ECUs architectures have also been explored to enable custom capabilities while maintaining complete AUTOSAR compliance [51]. Automotive suppliers and vendors have also explored hybrid FPGA-based ECUs for improved functional consolidation and reliability [19]. Specifically for network security, ML-based anomaly detection (bit-timing [52], transmission timing [53]) models, probabilistic attack models [54] and generalised multi-attack detection models [37], [38] have leveraged FPGA platforms for improved latency and detection performance. With the rising computational complexity in vehicles, it is expected that hybrid FPGA based ECUs will gain more traction for complex vehicular functions. Our approach aims at a integrated IDS-ECU architecture which consolidates the intrusion detection function through dedicated slave accelerator(s) with the primary ECU function on the same chip/package, mimicking an off-the-shelf ECU architecture. The accelerators will execute our QMLP models directly off received CAN messages from the network in full isolation and we show that such integration offers unique benefits in terms of latency, energy efficiency and detection accuracy across multiple attack vectors.

III. System Architecture

A. IDS-ECU Architecture

Figure 2 shows the proposed ECU architecture of the IDS-enabled ECU on a Xilinx Zynq Ultrascale+ device. The Zynq device integrate 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 with the PS using high or low performance Advaned eXtensible Interface (AXI) ports and can be accessed as memory mapped devices from the software application on the processor. In our proposed ECU architecture, standard ECU function(s) are mapped as software tasks onto one or more of the ARM cores on the PS, on top of a standard operating system like Linux or a real-time operating system. The operating system provides relevant drivers and APIs for accessing the PS peripherals and the PL accelerators, abstracting away low-level details of these blocks to create an

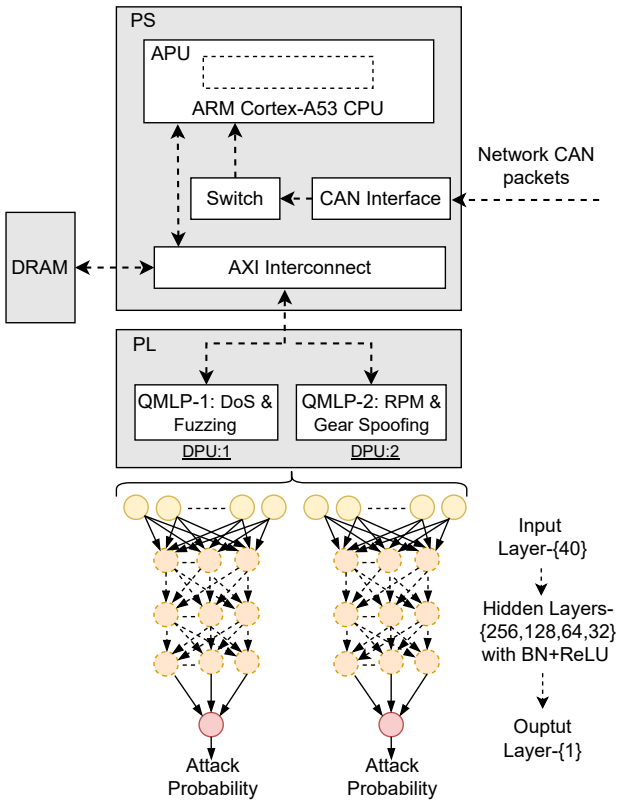

Fig. 2: Proposed system architecture of the IDS-ECU with the quantised MLP accelerators on the PL part of the FPGA.

AUTOSAR compliant architecture [51]. We propose to use the integrated CAN interface on the PS to handle the interfacing of ECU to the CAN bus, as shown in Figure 2.

The PL instantiates the Xilinx DPU [49] block as our dedicated ML accelerator which can be configured and controlled using the Vitis AI Runtime (VART) APIs from the PS. The standard Vitis-AI flow is used to automatically generate the DPU IP along with wrappers, interconnects and the runtime elements from our high-level ML model described in TensorFlow (see Sec. III-B). When the CAN interface receives a valid CAN packet, 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' specification. The isolated IDS task adds the relevant information from the CAN packet to the input feature buffer and uses the integrated Vitis-AI Runtime (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. We integrate two DPU cores which execute in 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.

B. MLP-based IDS

To determine the best ML model, we profiled different network architectures with varying complexity to find a baseline model that offers high inference accuracy at minimal complexity. We observed that while CNNs were effective for classification (with CAN IDs as input features), the MLP design provided more accurate detection (accuracy, false positives, and false negatives) when using the entire message contents at much lower computational complexity, and was hence chosen as our choice of architecture. A concatenation of $n = 4$ successive messages (CAN IDs + payload) is also chosen as the input feature based on this testing. Our model exploration was also guided by the network layers supported by the Vitis-AI framework. We used the open Car Hacking dataset for Intrusion Detection for our profiling, training, and validation [10].

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 (TF) using standard TF functions and nodes.

C. Dataset and Training

As mentioned above, we use the open Car Hacking dataset for training our model and to test its performance [55]. 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 II, showing the CAN ID (ID field), control field [Data Length Code (DLC) field] and the actual data segment (Data field). We split the dataset 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 the model on the validation set during training ensures that it is not over-fitting on any of the attacks. We pre-process the dataset prior to training to mimic the dataflow 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, 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.

To train the model, we used adam optimizer 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 [56]. For the first MLP, we first train the model on DoS attack and ensure optimal performance; subsequently, this trained model is trained 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.

TABLE II: An extract from the open Car hacking dataset which is used for our 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
...			

D. Mapping to DPU and Integration

The trained checkpoint files in full floating-point representation are 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. We further fine-tune the quantised model through a quantisation-aware optimisation step which improved the inference accuracy of the quantised model, particularly for the first MLP detecting DoS & Fuzzing attacks. Once optimized, the Vitis flow compiles the model to a DPU-executable 'xmodel' file, which incorporates the (quantised) weights, biases, 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 Linux operating system. Each DPU uses three master and one slave AXI interface to move instructions and data (inputs/weights/biases) between the PS and its internal memory to help maximise the peak performance. For this implementation, two instances of the B512 DPU configuration is deployed to minimise the resource overhead and power consumption. The Vivado flow generates the boot image for the Zynq device for booting from an external SD card with a standard Linux kernel.

IV. Deployment and Experimental Results

For training, we use the floating point variants of both models and use an Nvidia A6000 GPU for accelerating the training and validation, prior to quantisation through Vitis-AI. Post quantisation, the lightweight models have 53,300 parameters each and are packed into the 'xmodel' executable format. For our test, we use a Zynq Ultrascale+ ZCU104 development board which features a XCZU7EV Ultrascale+ device with quad-core ARM A53 cores and dual-core ARM R5 cores on the PS. 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 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 its execution and are configured to run concurrently for all our tests at startup. The Nvidia A6000 GPU has an base operating frequency of 1350 MHz.

We quantify the accuracy of inference by evaluating precision, recall, F1 rates as well as the false positive and false negative rates (FPR and FNR respectively) for our pre-quantised and quantised variants (on GPU and FPGA respectively). The processing latency and power consumption for performing IDS on each incoming CAN message on the ECU (using the DPU accelerators) is 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 are compared against state-of-the-art IDSs/IPSs proposed in the literature. In case of schemes where inference is performed on a block of CAN messages, we use these metrics along with the block size for the comparison. We also compare our active power consumption against ML-IDS approaches in literature where power consumption has been reported.

A. Accuracy

To test the functional correctness of the models, we first compare the pre- & post-quantisation inference performance of both models. We see a drop 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 III. Table IV shows the performance of QMLP-2 on the spoofing attacks for which we see no such degradation.

We compare the inference performance of our QMLPs integrated within the ECU against the state-of-the-art IDSs and IPSs proposed in the literature: GIDS [9], DCNN [10], MLIDS [14], NovelADS [11], TCAN-IDS [12], IForest [39], MTH-IDS [41], GRU [13] and Rec-CNN [40]. F1 score, precision, and recall metrics are used for comparison since most models report the inference performance using these metrics. The detailed results are tabulated in tables V and VI for DoS-Fuzzy attacks and Spoofing attacks respectively. Both MTH-IDS [41] and Rec-CNN [40] report an average accuracy, precision & recall of 99.9% across all attacks which is identical to our model on each of the three metrics. In GRU [13], authors report average precision, recall and F1 score values of 99.95%, 99.31% & 99.63% respectively for the spoofing attacks. Our model achieves the same (MTH-IDS & Rec-CNN) or better (GRU) results on all three metrics for these attacks.

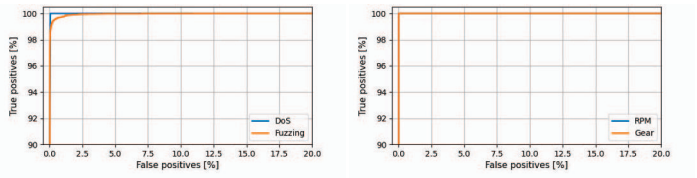

Fig. 3: ROC curve of the MA-QMLPs on the four attacks.

From table V, it can be observed that in case of the *DoS* attack, our approach offers identical performance to DCNN, MLIDS, NovelADS, TCAN-IDS and GRU models (F1 score). For *fuzzing attack*, our approach offers identical inference accuracy compared to DCNN, MLIDS, NovelADS, TCAN-IDS and performs better than iForest and GRU models by 2.3% and 0.5%, respectively (F1 score). Similarly, from table VI, it can be observed that our approach achieves $\geq$ 99.95% performance across both spoofing attacks across all parameters, and also perform 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 hence require multiple models to be deployed concurrently to detect all the attack vectors [10]–[13], [40]. In contrast, our 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 VII & VIII captures the confusion matrix of our QMLP models on the four attacks across the test vectors, showing the raw classification numbers achieved in each case. The misclassifications in our 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 shows the receiver operating characteristic (ROC) curves of the two QMLP models across the test vectors indicating that our classifier performs well for all discrimination thresholds. From the ROC curve, we can also observe that the area under curve (AUC) values achieved is $\geq$ 0.99 for all four attacks, showing that the model has a high probability of correctly distinguishing between normal and attack messages.

B. Inference latency

We quantify the per-message processing latency to show the potential of a tightly coupled IDS with ECU functionality to perform IDS on the message as soon as they arrive. The latency measure includes the time for move data from the 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

TABLE III: 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 IV: 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

TABLE V: Accuracy metric comparison (%) of our quantised FPGA accelerator (QMLP-1) against the reported literature on the DoS and Fuzzing attacks.

Attack	**Model**	**Precision**	**Recall**	**F1**	**FNR**
DoS	GIDS [9]	-	99.9	-	-
	DCNN [10]	100	99.89	99.95	0.13
	MLIDS [14]	99.9	100	99.9	-
	NovelADS [11]	99.97	99.91	99.94	-
	TCAN-IDS [12]	100	99.97	99.98	-
	iForest [39]	-	-	-	-
	GRU [13]	99.93	99.91	99.92	-
	QMLP-1 (DPU)	99.92	100	99.96	0
Fuzzy	GIDS [9]	-	98.7	-	-
	DCNN [10]	99.95	99.65	99.80	0.5
	MLIDS [14]	99.9	99.9	99.9	-
	NovelADS [11]	99.99	100	100	-
	TCAN-IDS [12]	99.96	99.89	99.22	-
	iForest [39]	95.07	99.93	97.44	-
	GRU [13]	99.32	99.13	99.22	-
	QMLP-1 (DPU)	99.86	99.67	99.76	0.33

in the literature in table IX, providing a comparison against approaches which 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 our ECU model achieve an end-to-end message processing latency of 0.24 ms for both the models since they are executed concurrently. This is a 2.3× speed up compared to MTH-IDS implemented on a dedicated IDS-ECU on a Raspberry-Pi 3 device. We also observe a significant speed-up compared to a loosely coupled GPU architecture performing inference (on the Nvidia A6000) as shown in table X; a portion of the higher 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,

TABLE VI: Accuracy metric comparison (%) of our quantised FPGA accelerator (QMLP-2) against the reported literature on the RPM and Gear attacks.

Attack	**Model**	**Precision**	**Recall**	**F1**	**FNR**
RPM	GIDS [9]	-	99.6	-	-
	DCNN [10]	99.9	99.9	99.9	0.05
	MLIDS [14]	100	100	100	-
	NovelADS [11]	99.9	99.9	99.9	-
	TCAN-IDS [12]	99.9	99.9	99.9	-
	iForest [39]	98.9	100	99.4	-
	QMLP-2 (DPU)	100	100	100	0
Gear	GIDS [9]	-	99.8	-	-
	DCNN [10]	99.9	99.8	99.9	0.11
	MLIDS [14]	100	100	100	-
	NovelADS [11]	99.8	99.9	99.9	-
	TCAN-IDS [12]	99.9	99.9	99.9	-
	iForest [39]	94.7	100	97.3	-
	QMLP-2 (DPU)	99.9	100	99.95	0

TABLE VII: Confusion matrix capturing the classification results of our QMLP-1 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 VIII: Confusion matrix capturing the classification results of our QMLP-2 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

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 IX.

We also quantify the cold-start setup time of IDS to determine the delay from power on to the model being ready to perform IDS. This is an important consideration since many attacks could be initiated during the startup phase of the network and ECUs. In the case of our test platform, we observe this cold-start latency as 8.9 seconds when averaged across multiple runs. It should be noted that our test platform

TABLE IX: Per-message latency comparison against other state-of-the-art IDSs reported in literature.

Models	Latency	Frames	Platform
GRU [13]	890 ms	5000 CAN frames	Jetson Xavier NX
MLIDS [14]	275 ms	per CAN frame	GTX Titan X
Rec-CNN [40]	117 ms	128 CAN frames	Jetson TX2
NovelADS [11]	128.7 ms	100 CAN frames	Jetson Nano
GIDS [9]	5.89 ms	64 CAN frames	GTX 1080
DCNN [10]	5 ms	29 CAN frames	Tesla K80
TCAN-IDS [12]	3.4 ms	64 CAN frames	Jetson AGX
MTH-IDS [41]	0.574 ms	per CAN frame	Raspberry Pi 3
QMLP-IDS (ours)	0.24 ms	per CAN frame	Zynq Ultrascale+

TABLE X: Per-message latency and measured power on the different platforms.

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

uses a slower boot device (SD card interface) and full Linux OS, whereas typical ECUs rely on non-volatile flash memory and real-time operating system, which could reduce the cold-start latency by 8× (with Linux and flash boot) or more.

C. Power consumption, PS/PL utilisation

We quantify the power consumption of our IDS-ECU 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, we observe an average consumption of 3.43 W 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 0.9mJ per inference. To model a loosely coupled GPU accelerator, we measure the power consumption of the quantised model using an A6000 GPU and the Nvidia 'nvtop' 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× reduction in power consumption as shown in table X and a 2172× better energy efficiency per inference making our approach a more efficient architecture for integrating IDS capabilities across ECUs. Among other reported results in the literature, our approach incurs a 2.6× reduction in power consumption when compared to the GRU [13] model on an Nvidia Jetson Xavier as a dedicated IDS node.

Finally, we quantify the overall hardware resource consumption of the IDS in table XI and observe 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. We also quantify the utilisation of the ARM cores 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). We believe that 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.

TABLE XI: 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%)

V. Conclusion

In this paper, we present an ECU architecture that consolidates IDS capabilities for CAN interface through a dedicated multi-core accelerator. The tightly coupled IDS-ECU architecture is capable of detecting DoS, Fuzzing & spoofing attacks using two lightweight quantised MLPs and deployed using two resource-efficient variants of Xilinx's DPU accelerator. Our lightweight MLP model matches or improves upon the detection accuracy metrics across multiple attack vectors using a single deployment, compared to the state-of-the-art models which require retraining and tuning to each attack vector. The proposed integration achieves a 2.3× reduction in per-message intrusion detection latency and a 2.6× reduction in power consumption compared to the state-of-the-art IDSs proposed in the literature that uses standalone and/or loosely coupled IDS integration on CPUs/GPUs. Our results show that the proposed architecture is best suited for the deployment of consolidated IDS capabilities within critical ECUs in a distributed fashion within the CAN network. In the future, we propose to expand our approach to integrate emerging vehicular networks like Automotive Ethernet and integration at network gateways for next-generation vehicular architectures.

VI. Acknowledgement

This research was supported by grants from NVIDIA and utilised NVIDIA RTX A6000 GPU.

References

[1] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking Tesla from wireless to CAN bus," *Briefing, Black Hat USA*, vol. 25, pp. 1–16, 2017.
[2] K. Iehira, H. Inoue, and K. Ishida, "Spoofing attack using bus-off attacks against a specific ECU of the CAN bus," in *Proc. IEEE Communications & Networking Conference (CCNC)*, pp. 1–4, IEEE, 2018.
[3] Z. Cai, A. Wang, W. Zhang, M. Gruffke, and H. Schweppe, "0-days & mitigations: Roadways to exploit and secure connected BMW cars," *Black Hat USA*, vol. 2019, p. 39, 2019.
[4] A. Greenberg, "After Jeep hack, Chrysler recalls 1.4 m vehicles for bug fix," *Wired*, 2015.
[5] R. B. GmbH, "CAN Specification, Version 2.0," 1991.
[6] U. E. Larson, D. K. Nilsson, and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *Proc. IEEE Intelligent Vehicles Symposium*, pp. 220–225, IEEE, 2008.
[7] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *Def Con*, vol. 21, no. 260-264, pp. 15–31, 2013.
[8] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi, "A language-based intrusion detection approach for automotive embedded networks," *International Journal of Embedded Systems*, 2018.

[9] E. Seo, H. M. Song, and H. K. Kim, "GIDS: GAN based intrusion detection system for in-vehicle network," in *Proc. Conf. on Privacy, Security and Trust (PST)*, pp. 1–6, IEEE, 2018.

[10] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Vehicular Communications*, vol. 21, p. 100198, 2020.

[11] K. Agrawal, T. Alladi, A. Agrawal, V. Chamola, and A. Benslimane, "NovelADS: A Novel Anomaly Detection System for Intra-Vehicular Networks," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[12] P. Cheng, K. Xu, S. Li, and M. Han, "TCAN-IDS: Intrusion Detection System for Internet of Vehicle Using Temporal Convolutional Attention Network," *Symmetry*, vol. 14, no. 2, p. 310, 2022.

[13] H. Ma, J. Cao, B. Mi, D. Huang, Y. Liu, and S. Li, "A GRU-Based Lightweight System for CAN Intrusion Detection in Real Time," *Security and Communication Networks*, vol. 2022, 2022.

[14] A. K. Desta, S. Ohira, I. Arai, and K. 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)*, pp. 1–7, IEEE, 2020.

[15] K. Cho, J. Kim, D. Y. Choi, Y. H. Yoon, J. H. Oh, S. E. Lee, *et al.*, "An FPGA-based ECU for remote reconfiguration in automotive systems," *Micromachines*, vol. 12, no. 11, p. 1309, 2021.

[16] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, "Reconfigurable computing in next-generation automotive networks," *IEEE Embedded Systems Letters*, vol. 5, no. 1, pp. 12–15, 2013.

[17] Zynq-Ultrascale-Plus-Product-Selection-Guide, "https://www.xilinx.com/content/dam/xilinx/support/documents/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf," 2018.

[18] K. Vipin, S. Shreejith, S. A. Fahmy, and A. Easwaran, "Mapping time-critical safety-critical cyber physical systems to hybrid FPGAs," in *Proc. Intl. Conf. on Cyber-Physical Systems, Networks, and Applications*, pp. 31–36, 2014.

[19] R. B. GmbH, *Engine Control Unit MS 6*. Robert Bosch GmbH,, 2015.

[20] F. Hartwich *et al.*, "CAN with flexible data-rate," in *Proc. iCC*, pp. 1–9, Citeseer, 2012.

[21] M. Bozdal, M. Samie, and I. Jennions, "A Survey on CAN Bus Protocol: Attacks, Challenges, and Potential Solutions," in *Proc. Intl. Conf. on Computing, Electronics Communications Engineering (iCCECE)*, pp. 201–205, 2018.

[22] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, "Practical DoS attacks on embedded networks in commercial vehicles," in *Proc. Intl Conference on Information Systems Security*, Springer, 2016.

[23] M. Enev, A. Takakuwa, K. Koscher, and T. Kohno, "Automobile Driver Fingerprinting.," *Proc. Priv. Enhancing Technol.*, vol. 2016, no. 1, pp. 34–50, 2016.

[24] K. Koscher, S. Savage, F. Roesner, S. Patel, T. Kohno, A. Czeskis, D. McCoy, B. Kantor, D. Anderson, H. Shacham, *et al.*, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symposium on Security and Privacy*, pp. 447–462, IEEE Computer Society, 2010.

[25] A. Palanca, E. Evenchick, F. Maggi, and S. 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*, pp. 185–206, Springer, 2017.

[26] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, and A. Mouzakitis, "Intrusion detection systems for intra-vehicle networks: A review," *IEEE Access*, vol. 7, pp. 21266–21289, 2019.

[27] T. P. Vuong, G. Loukas, and D. Gan, "Performance evaluation of cyber-physical intrusion detection on a robotic vehicle," in *Proc. Intl. Conf. on Computer and Information Technology*, pp. 2106–2113, IEEE, 2015.

[28] S. N. Narayanan, S. Mittal, and A. Joshi, "Using data analytics to detect anomalous states in vehicles," *arXiv preprint arXiv:1512.08048*, 2015.

[29] M. Weber, S. Klug, E. Sax, and B. Zimmer, "Embedded hybrid anomaly detection for automotive CAN communication," in *Proc. European Congress on Embedded Real Time Software and Systems*, 2018.

[30] D. K. Vasistha, *Detecting anomalies in controller area network for automobiles*. PhD thesis, 2017.

[31] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 911–927, 2016.

[32] K.-T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1109–1123, 2017.

[33] H. Lee, S. H. Jeong, and H. K. 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)*, pp. 57–5709, IEEE, 2017.

[34] W. Wu, Y. Huang, R. Kurachi, G. Zeng, G. Xie, R. Li, and K. Li, "Sliding window optimized information entropy analysis method for intrusion detection on in-vehicle networks," *IEEE Access*, vol. 6, 2018.

[35] A. Alshammari, M. A. Zohdy, D. Debnath, and G. Corser, "Classification approach for intrusion detection in vehicle systems," *Wireless Engineering and Technology*, vol. 9, no. 4, pp. 79–94, 2018.

[36] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, "Tree-based intelligent intrusion detection system in internet of vehicles," in *2019 IEEE global communications conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[37] S. Khandelwal, E. Wadhwa, and S. Shreejith, "Deep Learning-based Embedded Intrusion Detection System for Automotive CAN," in *Proc. Intl. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 88–92, IEEE, 2022.

[38] S. Khandelwal and S. Shreejith, "A Lightweight Multi-Attack CAN Intrusion Detection System on Hybrid FPGAs," in *Proc. Intl. Conf. on Field Programmable Logic and Applications (FPL)*, pp. 425–429, IEEE, 2022.

[39] P. F. De Araujo-Filho, A. J. Pinheiro, G. Kaddoum, D. R. Campelo, and F. L. Soares, "An Efficient Intrusion Prevention System for CAN: Hindering Cyber-Attacks with a Low-Cost Platform," *IEEE Access*, vol. 9, pp. 166855–166869, 2021.

[40] A. K. Desta, S. Ohira, I. Arai, and K. Fujikawa, "Rec-CNN: In-vehicle networks intrusion detection using convolutional neural networks trained on recurrence plots," *Vehicular Communications*, 2022.

[41] L. Yang, A. Moubayed, and A. Shami, "MTH-IDS: A Multitiered Hybrid Intrusion Detection System for Internet of Vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 616–632, 2021.

[42] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in *Proc. Design Automation Conference*, pp. 1–6, 2017.

[43] "A survey and taxonomy of FPGA-based deep learning accelerators," *Journal of Systems Architecture*, vol. 98, pp. 331–345, 2019.

[44] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "DLAU: A scalable deep learning accelerator unit on FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 513–517, 2016.

[45] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J. K. Kim, A. Mishra, and H. Esmaeilzadeh, "Dnnweaver: From high-level deep network models to FPGA acceleration," in *The Workshop on Cognitive Architectures*, 2016.

[46] E. Wang, J. J. Davis, P. Y. Cheung, and G. A. Constantinides, "LUTNet: Rethinking inference in FPGA soft logic," in *Proc. Intl. Symposium on Field-Programmable Custom Computing Machines*, IEEE, 2019.

[47] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proc. Intl. Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 65–74, 2017.

[48] Xilinx, "Vitis AI User Guide," 2021.

[49] Xilinx, "Zynq DPU v3.2," 2020.

[50] S. Shreejith and S. A. Fahmy, "Smart network interfaces for advanced automotive applications," *IEEE Micro*, vol. 38, no. 2, pp. 72–80, 2018.

[51] F. Fons and M. Fons, "FPGA-based automotive ECU design addresses AUTOSAR and ISO 26262 standards," *Xcell journal*, vol. 78, p. 20, 2012.

[52] J. Zhou, P. Joshi, H. Zeng, and R. Li, "Btmonitor: Bit-time-based intrusion detection and attacker identification in controller area network," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 6, pp. 1–23, 2019.

[53] Y. Yang, Z. Duan, and M. Tehranipoor, "Identify a Spoofing Attack on an In-Vehicle CAN Bus Based on the Deep Features of an ECU Fingerprint Signal," *Smart Cities*, vol. 3, no. 1, pp. 17–30, 2020.

[54] M. Casillo, S. Coppola, M. De Santo, F. Pascale, and E. Santonicola, "Embedded Intrusion Detection System for Detecting Attacks over CAN-BUS," in *Proc. Intl. Conf. on System Reliability and Safety (ICSRS)*, pp. 136–141, 2019.

[55] CAR Hacking Dataset, "https://ocslab.hksecurity.net/datasets/can-intrusion-dataset," 2020.

[56] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *arXiv preprint arXiv:1802.04680*, 2018.