

SecCAN: An Extended CAN Controller with Embedded Intrusion Detection

Shashwat Khandelwal & Shanker Shreejith
Reconfigurable Computing Systems Lab, Electronic & Electrical Engineering
Trinity College Dublin, Ireland
Email: {khandels,shankers}@tcd.ie

Abstract—Recent research has highlighted the vulnerability of in-vehicle network protocols such as controller area networks (CAN) and proposed machine learning-based intrusion detection systems (IDSs) as an effective mitigation technique. However, their efficient integration into vehicular architecture is non-trivial, with existing methods relying on electronic control units (ECUs)-coupled IDS accelerators or dedicated ECUs as IDS accelerators. Here, initiating IDS requires complete reception of a CAN message from the controller, incurring data movement and software overheads. In this paper, we present SecCAN, a novel CAN controller architecture that embeds IDS capability within the datapath of the controller. This integration allows IDS to tap messages directly from the bus, removing overheads incurred by existing ML-based IDSs. A custom-quantised machine-learning accelerator is developed as the IDS engine and embedded into SecCAN's receive data path, with optimisations to overlap the IDS inference with the protocol's reception window. We implement SecCAN on AMD XCZU7EV FPGA to quantify its performance and benefits in hardware, using multiple attack datasets. We show that SecCAN can completely hide the IDS latency within the CAN reception window for all CAN packet sizes and detect multiple attacks with state-of-the-art accuracy with zero software overheads on the ECU and low energy overhead (73.7 μ J per message) for IDS inference. Also, SecCAN incurs limited resource overhead compared to a standard CAN controller (< 30% LUT, < 1% FF), making it ideally suited for automotive deployment.

Index Terms—Smart network controllers, Intrusion Detection Systems, Quantised Neural Networks, Multi-layer Perceptrons

I. INTRODUCTION AND BACKGROUND

Most high-end vehicles today integrate over 50 electronic computing units (ECUs) interconnected through different network standards for incorporating safety-critical, comfort and automation capabilities in a cost and energy-efficient manner. CAN (and its variants) continue to be the most widely used network protocol in automotive electric/electronic systems owing to their low cost, flexibility, and robustness [1]. As a broadcast-based shared-bus protocol with minimal overhead, CAN has no built-in scheme for securing message exchanges over the network or providing sender/receiver authentication. Thus, any malicious node accessing the physical network can easily observe, decode and tamper CAN messages [2].

To mitigate these vulnerabilities, researchers have explored methods to detect and restrict malicious actors using intrusion detection and intrusion prevention systems (IDSs and IPSs respectively) [3]. Compared to rule-based and entropy-based IDS systems, the generalisable and scalable nature of ML-based IDS has made them ideally suited for embedded intrusion detection in vehicular ECUs [4], [5], [6], [7], [8]. However, the computational complexity of ML-based IDS is prohibitive to software deployment on the ECU as the IDS task can consume valuable time and resources required for the safety-critical and real-time functions on the ECU. Hence, most practical IDS integration schemes in the literature rely on complete ECUs dedicated to IDS (GPUs, FPGAs, or microcontrollers) or through a dedicated accelerator attached to the ECU. Figure 1 captures the different integration strategies proposed in the literature. Case 1 in figure 1 shows the IDS integrated as

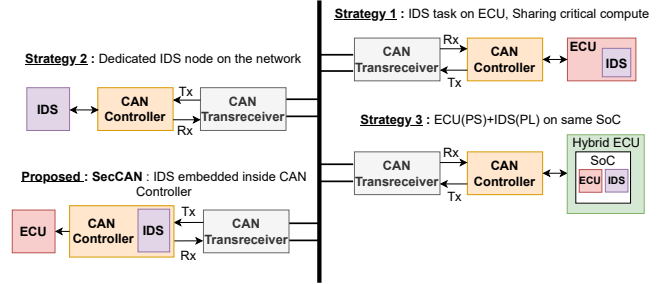


Fig. 1: The figure illustrates conventional integration strategies for CAN IDSs reported in the literature, and the proposed case for embedding IDS within the controller.

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 [9]. Case 3 shows a coupled accelerator, where the ECU offloads IDS to the accelerator on the same SoC once the packet is received from the CAN controller. In all the above cases, the CAN message has to be completely received by the controller and subsequently read by the coupled ECU/processing element before the ML model can perform IDS checks.

In this letter, we propose SecCAN, an extended CAN controller that integrates an IDS accelerator into its receive side datapath, as shown in figure 1 case 4. This critical design choice allows the IDS accelerator to directly extract CAN bus data (ID and payload) from within the receive path of the CAN controller for overlapping IDS execution with the current packet's reception. Complementing this with a compact 4-bit quantised multi-layer perceptron (QMLP) IDS model deployed as an unrolled dataflow accelerator, SecCAN completely hides the latency of IDS within the reception window of the current CAN frame. This is a departure from existing ML-based IDS solutions in the research literature, where IDS execution is triggered after the message is transferred to the ECU from the CAN controller. We evaluate SecCAN controller on a Zynq Ultrascale+ platform with the ARM cores on the Zynq device acting as the coupled ECU. The IDS performance of the SecCAN controller is evaluated by replaying messages from multiple datasets on the test platform. The key contributions of the letter are as follows:

- We introduce a novel CAN controller architecture (SecCAN) 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 two attack datasets [7], [10] and achieved state-of-the-art binary classification accuracy

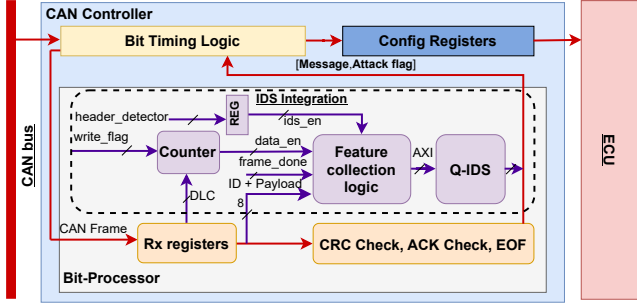


Fig. 2: The figure illustrates the integration of the IDS within the CAN controller. The red arrows show the standard datapath and the purple arrows indicate the augmented path for IDS.

of 99.993% and 99.966% (average) across multiple attacks.

- We implement the SecCAN controller on an AMD Zynq XCZU7EV FPGA 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 allows 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 [7], [11], [12], [13]. To the best of our knowledge, this work presents the first integration of an IDS operating in parallel with the CAN controller’s datapath. We further openly release the implementation of SecCAN to foster further research into smart network interfaces at <https://github.com/RCSL-TC/SecCAN>.

II. SECCAN ARCHITECTURE

A. Extending 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 our work, we utilise an open-source register transfer level (RTL) implementation of the CAN controller described in Verilog as our baseline controller [14]. The CAN protocol is implemented in the baseline CAN controller through the following modules: *config_registers*, *bit_timing_logic* and *bit_processing*.

The *config_registers* module implements the AXI interface to the ECU and incorporates a set of registers and frame buffers to capture protocol configuration, status signals and transmit/received messages. The memory-mapped registers within the *config_registers* module define command/control functions for other modules as well as error-tracking and status signals. At startup, the ECU configures the protocol registers to set operating parameters such as bit-level timing (sampling windows, clock prescaler configuration), interrupt configuration, and message filtering for the receive path, among others. Once active, a software task in the ECU writes up to 8 bytes of data to be sent as a CAN frame into the transmit buffer, and the controller attempts to transmit the payload based on the configured 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. Any frame to be passed upstream is written to the receive buffer and an interrupt signal is generated (if not masked), causing the ECU to read the received frame.

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. We extend the datapath within the *bit_processing* module to embed our 4-bit quantised MLP as the IDS (Q-IDS) within the CAN IP, as shown in figure 2. The extension brings together a set of control signals and the decoded byte from the controller’s modules to the feature collection logic that generates the input features for our IDS. The Q-IDS IP is an AXI stream accelerator of our quantised MLP generated using AMD’s FINN toolchain, starting from our high-level Python model. We apply selective unrolling and dataflow optimisations to optimise throughput and latency in the generated IP.

The bit processing operation begins when a header is detected on the bus and is indicated by the *header_detector* signal, marking a possible start of a new message on the CAN bus. The IDS control logic asserts *ids_en* to enable the feature collection logic and the counter starts tracking the number of data bytes written to the receive data register(s) (and also to our feature collector FIFO through the datapath extension) by monitoring the *write_flag* signal. When the counter value matches the data length code (DLC) of the current CAN frame, the frame header and message are available in the feature collection logic’s FIFO, and the *data_en* signal is asserted to begin the IDS operation. The new input feature vector comprising (CAN ID + Payload) of current and previous messages is subsequently transferred to the IDS IP. The feature collection logic zero pads smaller-sized payloads to 8 bytes to generate a uniform feature size for the IDS. This allows us to replay the bus messages as-is during our evaluation without pre-processing the dataset, faithfully replicating an in-vehicle scenario. Once the feature vector is transferred, the IDS processes the frame for potential attacks and asserts the *ids_output_ready* signal on completion. This operation overlaps with the CAN protocol checks, where the bit-processor validates the CRC and error flags of the received frame, and waits for the end-of-frame signal before transferring the valid CAN message to the receive buffer (in *config_registers* module). The IDS output value is wired out of the *bit_processing* module to the *top* module, where a custom multiplexing logic appends the IDS output to the received CAN message as it is transferred to the receive buffer and subsequently read by the ECU. Figure 3 illustrates the SecCAN operation using a waveform, highlighting how the IDS operations overlap with the CAN signalling checks (CRC, error flagging bit times). Our optimised IDS implementation completes the inference within this window, thus hiding this latency from the ECU.

B. Model & Dataset for training and testing

To arrive at our final QNN model’s architecture, we explored different configurations with varying complexity (number of layers and number of neurons in each layer) to find a model that offers high inference accuracy at minimal complexity. We use CAN ID + Payload information from each CAN frame as the input feature for the model to perform binary classification. We arrived at a 4-bit (weights/activations) quantised multilayer perceptron model (MLP) as the chosen configuration for the IDS, which provided the best validation accuracy during the training process. The QNN model is trained using *brevitas*,

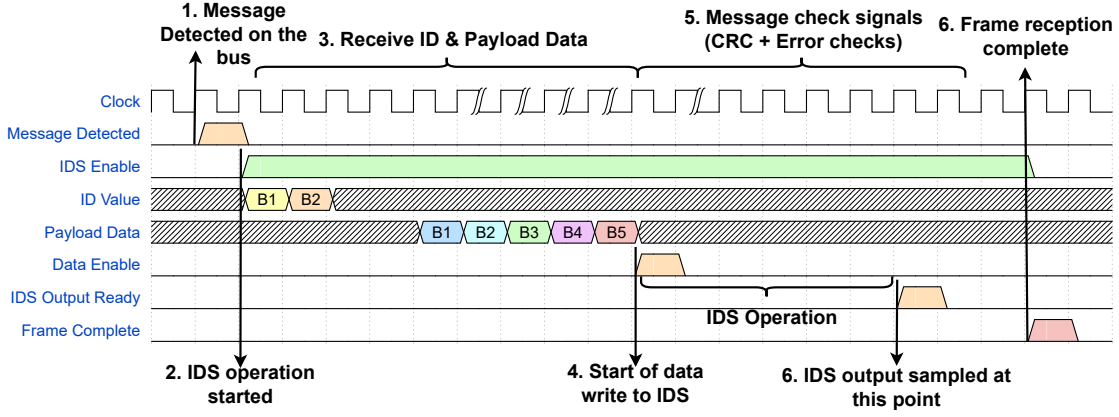


Fig. 3: The waveform shows the signalling within the SecCAN controller for a 5-byte CAN message from the bus. The IDS operation is overlapped with the protocol checks on the bus and is completed before the frame is ready to be read by the ECU.

which is an open-source quantisation-aware training (QAT) library for training neural networks [15]. The 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 to prevent overfitting during the training phase. The sigmoid function at the output denotes the probability of the current message being benign or malicious.

We used the open Car Hacking & survival analysis dataset for training and testing the model [7], [10]. The datasets contain CAN bus data acquired via the OBD port in an actual vehicle, with attack messages injected in real-time, compiled into a labelled set of normal and attack messages. We split each dataset into 75:15:10 proportions for training, validation, and testing.

III. EVALUATION & RESULTS

We generate the hardware configuration of the model 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 maximum bitrate of 1 Mbps, with a CAN clock frequency of 16 MHz. We evaluate the 4b-QMLP IDS model's accuracy in detecting active injection attacks and compare it 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 at the cost of higher resource and energy consumption (see sec. III-C). Additionally, we measure the inference latency in hardware and analyse the trade-offs in resource and energy utilisation that the controller incurs from this integration. For measurements, we replay the attack messages either directly from the ARM processor core (on the Zynq platform for testing accuracy) or from a BRAM that replays CAN traffic data (for quantifying latency and power).

A. Accuracy

We quantify the accuracy of the 4b-QMLP model in terms of precision, recall & F1 scores and compare it against state-of-the-art works in the literature, and across multiple datasets. Our test split from the Car Hacking dataset incorporates 75,000 messages each from the DoS & fuzzing attack datasets. The model achieves a binary classification accuracy of 99.993% (11 misclassifications for 150,000 test messages). Our test split from the survival analysis dataset

combines flooding, fuzzing, and malfunction attack vectors. On this test set, our model achieves a classification accuracy of 99.966% (25 misclassifications for 75,000 test messages). Table I presents a comparison of our model with others proposed in the literature for the two attack datasets, highlighting that our approach matches or surpasses the performance of competing methods.

B. Detection Latency

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} can be determined as $T_{max} = T_{frame_done} - T_{header_detected}$, where T_{frame_done} & $T_{header_detected}$ marks the time at completion of frame reception and valid header reception respectively. Since our 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 $37.376 \mu s$; hence, for real-time detection, the upper limit on IDS latency (T_{IDS}) must satisfy $T_{IDS} = T_{frame_done} - T_{data_en} < 37.376 \mu s$. This relation holds for any data length at 1 Mbps

TABLE I: Inference accuracy (%) of SecCAN compared to competing IDS schemes on both datasets.

| Attack | Model | Precision | Recall | F1 | % FNR |
|---------------------------------------|--------------------------|-----------|--------|-------|-------|
| DoS (Car Hacking) | DCNN [7] | 100 | 99.89 | 99.95 | 0.13 |
| | NovelADS [13] | 99.97 | 99.91 | 99.94 | - |
| | TCAN-IDS [16] | 100 | 99.97 | 99.98 | - |
| | GRU [12] | 99.93 | 99.91 | 99.92 | - |
| | iForest [9] | 95.07 | 99.93 | 97.44 | - |
| | 4b-QMLP in SecCAN | 99.99 | 99.98 | 99.98 | 0.02 |
| Fuzzing (Car Hacking) | DCNN [7] | 99.95 | 99.65 | 99.80 | 0.5 |
| | NovelADS [13] | 99.99 | 100 | 100 | - |
| | TCAN-IDS [16] | 99.96 | 99.89 | 99.22 | - |
| | GRU [12] | 99.32 | 99.13 | 99.22 | - |
| | iForest [9] | 95.07 | 99.93 | 97.44 | - |
| | 4b-QMLP in SecCAN | 99.99 | 99.97 | 99.98 | 0.03 |
| Flooding (Survival Analysis) | XGBoost [17] | 100 | 90 | 94.74 | - |
| | G-IDCS [18] | 99.72 | 99.72 | 99.72 | - |
| | LSTM [19] | - | 100 | 100 | 0 |
| | 4b-QMLP in SecCAN | 100 | 100 | 100 | 0 |
| Fuzzing (Survival Analysis) | XGBoost [17] | 99.98 | 99.08 | 99.53 | - |
| | G-IDCS [18] | 100 | 100 | 100 | 0 |
| | LSTM [19] | - | 99.95 | 99.96 | 0.05 |
| | 4b-QMLP in SecCAN | 99.98 | 99.50 | 99.74 | 0.5 |
| Malfunction (Survival Analysis) | XGBoost [17] | 99.92 | 100 | 99.96 | - |
| | G-IDCS [18] | 100 | 99.64 | 99.82 | - |
| | LSTM [19] | - | 100 | 100 | 0 |
| | 4b-QMLP in SecCAN | 100 | 100 | 100 | 0 |

TABLE II: Resource utilisation on the FPGA for the standard CAN controller (CAN-NC) & our 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 III: Comparison of SecCAN’s energy consumption per inference with other IDS approaches proposed in the literature.

| Model | Platform | Energy consumption |
|--------------|------------------|--------------------|
| GRU [12] | Jetson Xavier NX | 1.77 mJ |
| QCAE [21] | Zynq Ultrascale+ | 2.09 mJ |
| MTH-IDS [11] | Raspberry Pi 3 | 1.29 mJ |
| iForest [9] | Raspberry Pi 4 | 390.6 μ J |
| SecCAN | Zynq Ultrascale+ | 73.7 μ J |

CAN bus configuration. From our evaluation (both from simulation and hardware measurements), we observe a detection latency of 36.5 μ s at 16MHz clock, allowing the SecCAN to tag each message with an attack or benign flag before the reception window of current message on the network ends. Further, we observe a 10.9 \times reduction in latency for our unrolled 4b-QMLP compared to an 8-bit implementation on a Jetson Xavier, even with reception and message transfer times excluded.

C. Resource & Energy Consumption

We report the resource consumption and energy numbers of the SecCAN controller, compared to the standard CAN controller in table II & III respectively. We can observe 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 as the controller. We further measure the per-message energy consumption of the IDS by monitoring the power rails on the ZCU104 board during IDS execution. Table III compares SecCAN’s energy consumption per inference against other works in the literature which have reported energy consumption. Our model consumes only 73.7 μ J per message on average, a 3.1 \times improvement over our 2-bit quantised coupled accelerator IDS [20], achieved primarily through lower operating speed and the Q-IDS IP optimisations. 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. Similarly, SecCAN consumes 24.1 \times and 28.4 \times lower energy per inference than a GRU-based IDS [12] and a convolutional autoencoder-model [21] on Jetson Xavier and Zynq Ultrascale+ platforms respectively. While other competing schemes such as MTH-IDS [11] and iForest [9] do not explicitly report their energy consumption, they are re-implemented on Raspberry Pi-3 and Pi-4 devices for comparison (as in the original article). In comparison to these, SecCAN’s energy consumption was found to be 17.6 \times and 5.3 \times lower per inference. It should be noted that energy measurement for SecCAN is for the extended CAN controller (SecCAN) as opposed to the IDS-only energy consumption reported and measured for competing schemes.

IV. CONCLUSION

In this letter, we explore a smart CAN controller architecture that integrates a light-weight 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. We believe that the SecCAN architecture is scalable and can pave the way towards smarter IDS approaches in existing and future automotive networks.

REFERENCES

- [1] S. C. HPL, “Introduction to the controller area network (CAN),” *Application Report SLOA101*, pp. 1–17, 2002.
- [2] S. Rajapaksha, H. Kalutarage, M. O. Al-Kadri, A. Petrovski, G. Madzudzo, and M. Cheah, “AI-based intrusion detection systems for in-vehicle networks: A survey,” *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–40, 2023.
- [3] R. Bhatia, V. Kumar, K. Serag, Z. B. Celik, M. Payer, and D. Xu, “Evading Voltage-Based Intrusion Detection on Automotive CAN,” in *NDSS*, 2021.
- [4] S. N. Narayanan, S. Mittal, and A. Joshi, “Using data analytics to detect anomalous states in vehicles,” *arXiv preprint arXiv:1512.08048*, 2015.
- [5] 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.
- [6] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, “Tree-based intelligent intrusion detection system in internet of vehicles,” in *Proc. Global Communications Conf. (GLOBECOM)*, pp. 1–6, IEEE, 2019.
- [7] 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.
- [8] S. Tariq, S. Lee, and S. S. Woo, “CANTransfer: transfer learning based intrusion detection on a controller area network using convolutional LSTM network,” in *Proc. ACM Sym. on Applied Computing*, pp. 1048–1055, 2020.
- [9] P. Freitas 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.
- [10] M. L. Han, B. I. Kwak, and H. K. Kim, “Anomaly intrusion detection method for vehicular networks based on survival analysis,” *Vehicular communications*, vol. 14, pp. 52–63, 2018.
- [11] 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.
- [12] 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*, 2022.
- [13] K. Agrawal, T. Alladi, A. Agrawal, V. Chamola, and A. Benslimane, “NovelADS: A Novel Anomaly Detection System for Intra-Vehicular Networks,” *IEEE Trans. on Intelligent Transportation Systems (ITS)*, vol. 23, no. 11, 2022.
- [14] Opencorecan, “https://opencores.org/projects/can,” 2017.
- [15] A. Pappalardo, “Xilinx/brevitas,” 2023.
- [16] 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.
- [17] A. Anjum, P. Agbaje, S. Hounsinnou, and H. Olufowobi, “In-Vehicle Network Anomaly Detection Using Extreme Gradient Boosting Machine,” in *Proc. Med. Conf. on Embedded Computing (MECO)*, pp. 1–6, 2022.
- [18] S. B. Park, H. J. Jo, and D. H. Lee, “G-ids: Graph-based intrusion detection and classification system for can protocol,” *IEEE Access*, vol. 11, pp. 39213–39227, 2023.
- [19] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, “LSTM-based Intrusion Detection System for In-Vehicle CAN Bus Communications,” *IEEE Access*, vol. 8, pp. 185489–185502, 2020.
- [20] S. Khandelwal and S. Shreejith, “Exploring Highly Quantised Neural Networks for Intrusion Detection in Automotive CAN,” in *Proc. Intl. Conf. on Field-Programmable Logic and Applications (FPL)*, pp. 235–241, 2023.
- [21] S. Khandelwal and S. Shreejith, “Real-time Zero-day Intrusion Detection System for Automotive Controller Area Network on FPGAs,” in *Proc. Intl. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 139–146, 2023.