HARDWARE-IN-THE-LOOP EMULATOR FOR IN-VEHICLE CONTROLLER AREA NETWORKS

Milestone Report for Research Project Module 5E1

Changhong Li
Trinity College Dublin
lic9@tcd.ie

April 2024

This report is submitted in part fulfilment for the assessment required in 5E1 Research Project. I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year. These are found in Parts II and III at http://www.tcd.ie/calendar.

The report documents project progress and discusses achievements so far, based on plans outlined in the interim report due in January 2024. The report outlines project challenges and plans for the remainder of the project.

1 A review of the Project Goals

The main goal of this project is to build an open reconfigurable CAN bus hardware-in-the-loop simulation system on FPGA to realize security simulation of the CAN bus communication system of multi-platform ECUs and implement automatic attack vector injection (such as Dos, Replay and Spoofing), perform status monitoring and data analysis on communication networks. To decrease the time cost of security test and model establishment with heterogeneous platforms, we provide a set of solutions to easily and quickly establish the deep neural networks and deploy them into the hardware environment through FINN to deal with the rapid changes and increasingly challenges in automotive communication security. The structure of the system is shown in the figure 1 below.

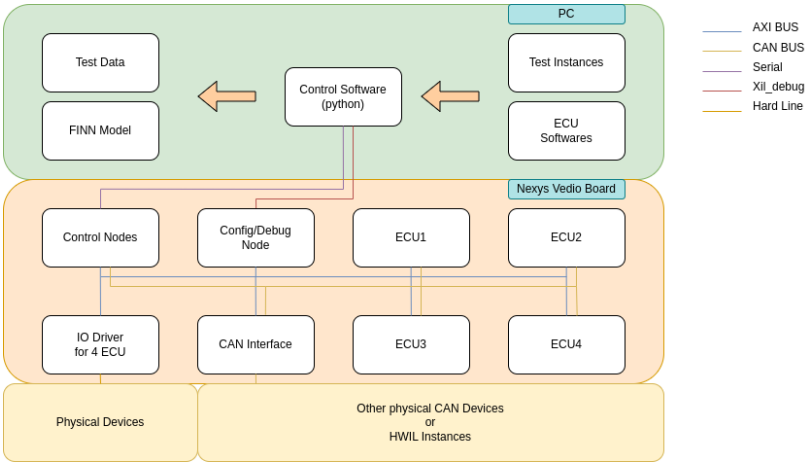

Figure 1: *System structure*

In recent years, with the development and progress of new energy and intelligent vehicles, in order to improve the intelligent performance of their products, optimize user experience and improve market competitiveness, vehicle-mounted intelligent devices have become more and more complex. This also means new challenges and higher requirements have been put forward for the security and reliability of the communication process. ECUs of the vehicle system is usually connected with CAN bus to control each sub-system domain. Around 1990, a number of automotive bus protocols emerged, such as Flex Ray, LIN, and CAN buses. Among them, the most widely used open bus in the world is the CAN bus proposed by Bosch[1][2]. Its efficiency and compatibility have been recognized by the industry, but CAN bus is easily exploited due to its communication characteristics that each node has same permission and no masters and slaves which make it possible to be attacked by injection. The losses caused by automotive cyber-attacks are often fatal and unacceptable. For example, the failure of car control and the stoppage of safety systems will cause significant harm to users and machines.

Industry-related fields have also made a lot of efforts in this regard, but usually network changes of a certain scale will lead to changes of the whole system, which may lead to the emergence of new vulnerabilities. Existing entropy-based analysis or neural network-based models due to environmental Changes in data can cause large changes in performance, which usually requires large-scale system testing and model re-establishment. The testing costs, time costs and even performance or security costs caused by such changes are still issues that need to be addressed.

From the analysis of security requirements and industry needs, this research is necessary and has certain application value.

2 Updates to previous work

In the interim report, we sorted out and summarized previous work from the aspects of hardware-in-the-loop simulation, attack injection, attack detection, and neural network attack detection. In the milestone report, we further supplement and update this content, especially the network anomaly detection and prevention section.

In DNN for attack detection, such as Song et al. [3] was the first to propose applying DCNN to vehicle network security research on IDS, using the ResNet architecture, which is the earliest framework applied to image processing. Shahroz et al. proposed to use long short-term memory model to detect attacks on CAN bus [4]. Shashwat and Shanker et al. summarized the current methods of using artificial intelligence to detect CAN bus attacks and found that CAN ID, Data Field and DLC were mainly used as input information to train the model, and different models and deployment methods were studied. [5]

AutoEncoder is a relatively mainstream anomaly detection technology that we have not mentioned before, it is an unsupervised model that learns the universal characteristics of features and eliminates redundant information by compressing the input feature vector to a lower dimension and reconstructing it. By reconstructing the characteristic data and checking the reconstructed characteristics, it is determined whether the target data is an attack vector. This method has been proposed in the industry and is an excellent general anomaly identification solution, but it still has certain shortcomings.

Explainability, the working principle of the neural network model is difficult to be understood or explained by humans[6]. For the detected anomalies, DNN can give its category and explain to a certain extent the reason why it is marked as an anomaly. However, the Explainability of AE model[7] is poor, it could only tell that the data is abnormal and the only reason is high loss, but can not tell why. Typically, in high-reliability automotive communications applications, explainability is necessary.

Accuracy, the main advantage of AE is its versatility. In the absence of a large amount of labeled data, AE will show better advantages than DNN to a certain extent. However, when there is sufficient labeled data, DNN can often provide better accuracy and performance. Our framework is officially able to provide this large amount of annotated data in line with flexible reconfigurable networks and conduct model training, which also solves the missing part of annotated data, which could be the hard part for the previous model-build works. Shashwat and Shank et al. summarized the current use of artificial intelligence detection among the CAN bus attack methods, the model training mainly uses the open source CAN bus message data set. We can quickly retrain and supplement the performance of the model based on the actual network structure to quickly adapt to the new network structure.

To sum up, based on the author's limited knowledge, different from the previously mentioned technical solutions, this method is relatively high-performance, flexible, scalable, and explainable

attack vector test and detection solution, which could better match industry requirements and address key issues in flexibility and high performance.

3 Milestone Review

In the interim report, 5 main milestones were set. These have been discussed with our supervisor at regular intervals and some alterations had to be made. Each milestone is now discussed in turn.

3.1 ECU simulation topology

ECU is the main control node and the smallest unit in the automotive control system. Each ECU node is connected using the CAN bus and has a control domain and completes its respective role, such as controlling local digital or analog quantities, signal input and output, or controlling power supply or motor drive system.

In order to simulate this working environment, we use a 4-core Micro-blaze controller on the FPGA and assign certain peripherals to them to simulate actual working conditions. At the same time, we also established a 4-core Risc-V controller topology network version. Although they use different architectures, their working modes and application download modes are consistent. Here we focus on describing the former one, the system structure is relatively flexible. A sufficient number of ECU topological networks can be built by increasing the number of soft IP cores or increasing the number of FPGAs physically connected. These ECUs could have different instruction sets, different functionalities, or equipped with different physical interfaces.

For facilitate simulation testing, we built a relatively simple automotive application topology network, and its network structure is shown in the figure below2. In this topological network, each ECU plays different roles.

- **ECU1: Control Engines, transmission unit and brake unit**
- **ECU2: Control Airbags and Steering Lights**
- **ECU3: Physical IO inputs like Collision detector, brake detector, Lights control**
- **ECU4: Control Front light and back light of the car**

In addition to its own functionality, each ECU will also perform communication connections according to a communication protocol similar to the figure below3. The format of this communication protocol refers to the train control protocol template, and is only used as an illustration of simple application.

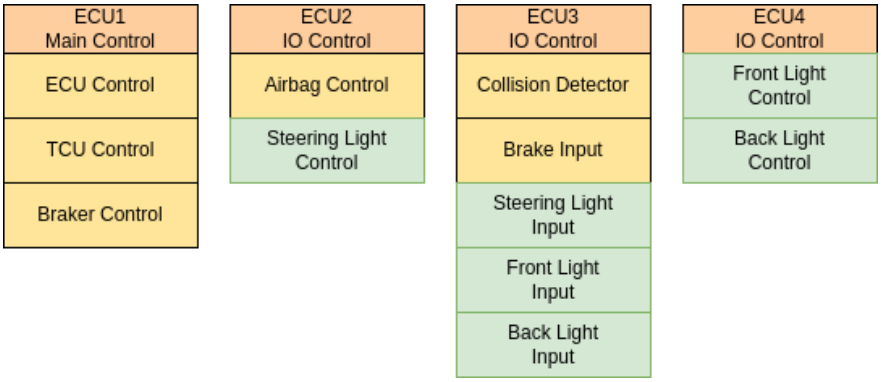

Figure 2: *CAN bus structure*

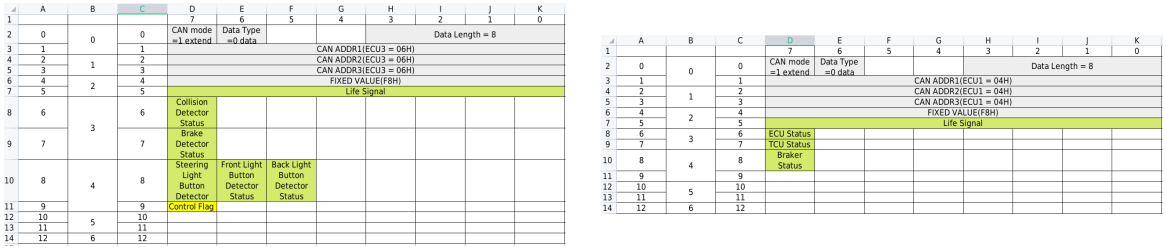

Figure 3: *Communication protocol*

Its basic logic is to simulate the ECU's peripheral devices through external physical IO, and by triggering the corresponding semaphore input, each ECU will make relevant actions. For example, when ECU3 detects a collision, ECU1 will block the engine and transmission unit, and ECU2 will enable the airbag to protect the passengers. When the passenger gives a turn light signal instruction through the physical switch, ECU3 accepts this instruction and sends it to the IO execution unit and activates the turn signal. At the same time, each ECU will periodically send its incremental life signal to announce its normal running status.

3.2 CAN controller IP core and virtual network

The CAN bus was first proposed by Bosch, although Xilinx provides a mature IP core of the AXI-CAN controller in its tool chain, this core belongs to the Logic expansion package which is not a free one and will cause problems for the later implementation of the project as unnecessary trouble and intellectual property expenses. Moreover, the expansion package only supports physical interface expansion and has insufficient support for onboard hardware-in-the-loop simulation application scenarios. Therefore, we use the open source CAN controller from Open-Core. The

source code of this CAN controller is derived from the early Bosch open source code. It has relatively complete functions and basic test cases.

We repackaged the interface of the IP and encapsulated the original interface into an AXI bus interface adapted to the on-chip controller. We established new test cases for the interface and conducted necessary tests on its performance and logic timing. The testing timing diagram of the controller is shown in the figure below4. At the same time, for the convenience of calling the control cores, we have also encapsulated a sample driver for it. Just like other AXI peripherals, it can be called with only simple initialization configuration and ready-made basic functions.

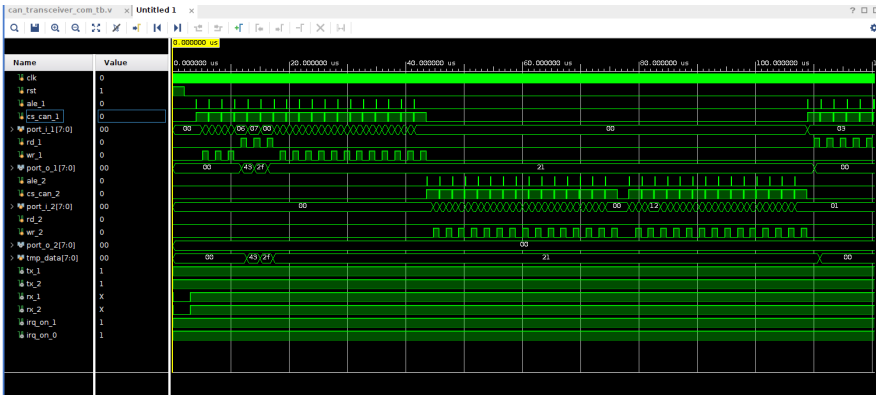

Figure 4: *IP test diagram*

Usually CAN bus communication at the physical level requires three important components, the logic controller, the CAN controller and the CAN transceiver. The CAN transceiver is used for level conversion and signal monitoring of send messages for data verification. In order to achieve compatibility, in the scenarios of with hardware and no physical hardware, we have also built a soft IP core of the transceiver, which can support bus control communication simulation completely on-chip in an environment without physical hardware.

3.3 Test and control software

In order to facilitate system testing and data collection and analysis, we used PYQT to build an operating host software. The control software is connected to the control node on the FPGA through serial communication, monitors the data on the CAN bus in real time and provides an interface for attack execution. The User interface is shown below5.

The main functions of the host computer are as follows:

- **Status Monitoring**

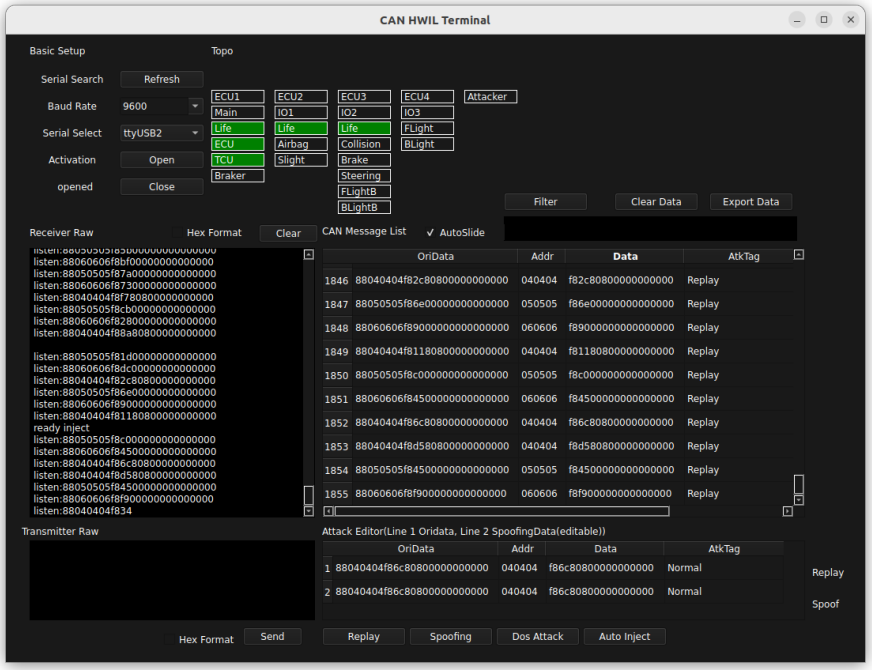

Figure 5: *User interface*

The host computer monitors the life signals and status data of each ECU node in real time and displays it on the interface in a visual form, allowing users to clearly and intuitively observe the operating status of each ECU and the impact on each node of the system before and after the attack. The host computer monitors the messages that occur on the CAN bus in real time, and performs a preliminary analysis of the message content, which is displayed on the interface in the form of a classified list for analysis and selection. It also supports the direct use of RAW messages to send and receive CAN buses which could make it is convenient for debugging and detection on the CAN Bus.

- **ECU program download**

  The program supports loading elf object code files, flexibly loads ECU applications onto each ECU core through the program, and supports remote control reset of these ECUs to facilitate testing of different versions of applications and automatically restore ECUs' status to facilitate multiple rounds of continuous testing to obtain more data. This function is based on automatically executed Tcl scripts.

- **Test data export**

  The program can directly export the message information that occurred during the test into

a data set format that can be used for FINN model training. Users can directly convert the network communication data in the current topology environment into a model training data set and use it for the attack detection model training.

- **Attack execution**

  Attack the target network according to the user's wishes, the specific content is elaborated in the next section.

PYQT is a framework adapted to multiple platforms. The host control software can be deployed on any terminal without special requirements. It only needs to install basic python and xilinx runtime environments, which makes its deployment very simple and requires no additional costs.

3.4 Attack injection

For the attack execution part, the system supports three basic attack forms as DoS attack, Spoofing attack and Replay attack.

As mentioned above, the monitoring system can check and capture real-time communication messages in the CAN bus network. Through the captured messages, the tester can choose to replay the message or modify it and send it as a spoofing message to attack. For example, the user can replay the control message sent by ECU3 so that its instructions are executed twice, or modify the content of its payload and change its control message to achieve the effect of the attack. These injection messages are sent to the control node and then sent to the CAN bus network.

When a user uses a DoS attack, the control node obtains the attack instructions and continuously injects high-priority meaningless messages into the bus network within a specified time to achieve the purpose of blocking network communication. The arbitration process of CAN bus communication mainly depends on the CAN message ID to determine its priority. Smaller CAN IDs usually have higher priorities. When we quickly and continuously inject the highest priority CAN bus messages into the bus, Other communication messages will be blocked, which will result in emergency control instructions being unable to be delivered. For example, if a collision signal triggers the airbag and this application is controlled by software, this process may be blocked and cause serious consequences. The DoS attack will also block the expression of the life signal of each node, making its life status unable to be checked by other nodes. During the attack, we can find that the life signal is lost and the airbag cannot be effectively triggered.

At the same time, we allow the software to execute automated attack injection scripts in a single time or in a loop. By writing basic information such as attack messages and attack types into csv files, the software can load the attack scripts and automatically launch attack tests to generate data sets.

3.5 Attack detection model

For the attack detection model, we use the multi-layer perceptual neural network, just like the model mentioned in Shashwat's work. The schematic diagram of the model structure is as follows6.

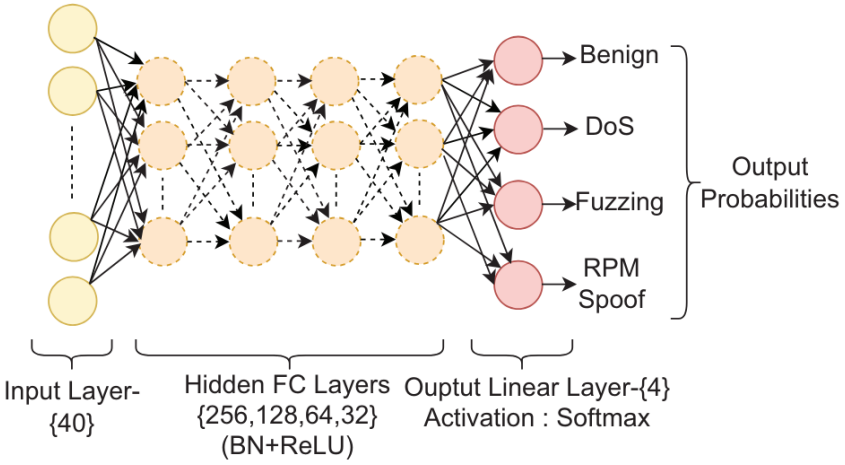

Figure 6: *Model Structure*

The input layer of the network is four consecutive CAN bus messages, whose content includes the ID, length, message content and other information of the bus message. Its data structure is shown in the figure below7. The output result is the model's confidence in the message type classification. By comparing the confidence model, the model can output the message type judgment.

If the model is deployed on FPGA, in addition to training the model, it must also go through the process of quantification and stitching. Since embedded devices such as FPGA and MCU are usually limited by hardware performance, it is impossible to fully achieve the accuracy achieved by the parameters of the original model. In order to achieve this trade-off, we used the Brevitas framework to quantize the multi-layer perceptual neural network model with 2 bits .

By using the FINN framework provided by Xilinx, this quantized neural network model is converted into a stitched IP core adapted to traditional FPGAs. There will be a certain loss in performance when stitching the IP core, but in the context of edge computing deployments, this sacrifice is worth it.

The stitched IP core has an AXI-Stream interface. We will use FIFO to feed real-time CAN bus data to the model, and use FIFO to receive the model's CAN message judgment results. Currently, the model has been established but has not been deployed on the FPGA, because the general stitched IP The workflow only supports the ZYNQ platform and proprietary platforms, but

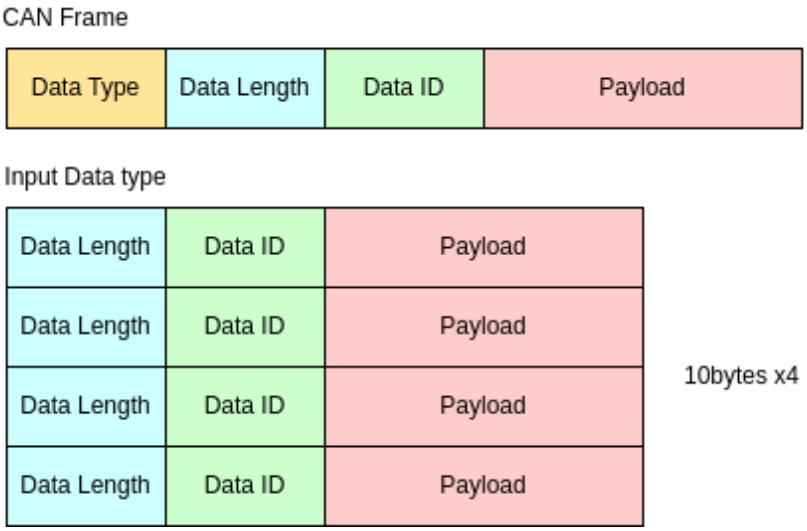

Figure 7: *Input Data Structure*

this cannot meet the requirements for universal adaptability. We must use another solution to convert it into an IP core that supports the traditional Artix platform. After the milestone reporting, we will deploy, test and perform performance tuning on the Stitch IP according to the established plan.

4 Final Comments

Based on the current milestone progress, we can consider the next main work content. The main work content in the next phase is model deployment, performance tuning, as well as overall system optimization and paper writing. The time list can be found in the appendixA Updated Gantt chart.

4.1 Model deployment

As mentioned previously, the general FINN model deployment workflow only supports deployment on ZYNQ and related platforms, and does not support FPGA deployment on traditional platforms. We will refer to the official Xilinx open source project FINN on the github and try to pass the quantified IP core through Specialized toolchains which could convert the model into stitching IP that can be adapted to the Artix platform. We have now completed the initial construction of the model and the read and write test of AXI-Stream by Microblaze. Next we will try to convert and deploy the model, and operate the FINN core through Microblaze to achieve real-time attack

detection and try to transmit the detection results back to the control software to achieve the purpose of real-time monitoring and performance monitoring.

4.2 Performance tuning

The current FINN model is a multi-layer perception recognition based on the combination of ID, data length, and load information. We will try to use network models with different structures, change their configurations, and then test, compare and optimize their comprehensive performance. These attempts may include modifying the network structure, the quantization parameters of the model, modifying the data type of the input layer, etc. In terms of model performance evaluation, we will compare the comprehensive parameters of different models under different working conditions, such as their accuracy and false alarm rate as known as the bench-marking.

At the same time, the current FINN model is trained based on mature CAN bus messages. Training the model under the conditions of existing data sets may not be suitable for new devices, new network topology environments, and the content of new bus messages, which may affect performance. If the network environment declines, we will use the data records obtained through the platform as a new data set, freeze some layers if necessary, and retrain the model using new data to improve the adaptability of the model in the new network environment and improve performance, which is also an effective content on the topology adaptability issue in the entire workflow.

4.3 System optimization

Regarding the robustness, reconfigurability and agility of the entire system, we will re-optimize and evaluate the system, and further improve the reliability of the system by optimizing the software algorithm, optimizing the model structure, and optimizing the human-computer interaction interface. If time permits, these optimizations may include but not limited to optimizing communication interfaces to improve the throughput of bus information monitoring, software enabling CAN bus nodes to improve their adaptability, configurable UI interfaces to avoid secondary development, and configurable communication protocols, automatically generate ECU control programs, etc.

4.4 Summary

The project is generally proceeding according to the existing plan, but there are certain deviations and changes. This is because there are relatively certain difficulties such as communication interface, the BSP program of the control core, and the deployment of FINN on atypical traditional FPGAs. However, we have alleviated these problems to a great extent by changing the control software communication form, calling the microblaze core and referring to the official FINN project to build stitching IP for any FPGA platform. So far, our results show that our testing framework can

run effectively, but we still have some work to complete on the deployment of stitched IP for the model and the performance optimization of the model. Even though our framework design is still at a relatively prototype stage, we are still able to write about our scientific journey so that others can learn from our experience. Given the changes in project milestones discussed, we hope to be able to complete the paper on time.

Bibliography

[1] Nicolas Navet, Yeqiong Song, Francoise Simonot-Lion, and Cédric Wilwert. Trends in automotive communication systems. *Proceedings of the IEEE*, 93(6):1204–1223, 2005.

[2] 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.

[3] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.

[4] Shahroz Tariq, Sangyup Lee, and Simon S Woo. Cantransfer: Transfer learning based intrusion detection on a controller area network using convolutional lstm network. In *Proceedings of the 35th annual ACM symposium on applied computing*, pages 1048–1055, 2020.

[5] 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.

[6] Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold, and Peter M Atkinson. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(5):e1424, 2021.

[7] Stephan Matzka. Explainable artificial intelligence for predictive maintenance applications. In *2020 third international conference on artificial intelligence for industries (ai4i)*, pages 69–74. IEEE, 2020.

A Gantt Chart