Georgia Institute of Technology

CS 4220/6235: RTES

Fall 2025

Project Checkpoint Report

Group: 19
Name(s): Phineas Giegengack
Project: Use of Machine Learning to Detect Attacks on CAN (Controller Area Network) in Embedded Microcontroller

Project Plan (Scope):

Introduction

The Controller Area Network (CAN) protocol is a specification which describes how many electronic control units (ECUs) broadcast data over a shared bus. CAN was derived in the 1980s for use in automobiles to reduce the complexity of the electrical design of vehicles with an ever-growing array of electronic sensors and actuators. CAN was an answer to difficulties that arose from designing point-to-point connections between every node that needed to communicate in a car, but CAN does not by default have authentication, which leaves it open to a variety of attacks, including Denial of Service (DoS), Fuzzing, and Spoofing.
In modern vehicles, and especially in autonomous vehicles, safety-critical systems may be connected via CAN, so the ability to detect intrusions onto this network is of utmost importance.

Many researchers have turned to machine learning to detect intrusions on the CAN bus, but relatively few have addressed the inherent limitations of applying machine learning in real-time and on limited hardware. For this project, I will aim to develop a machine learning pipeline using the Autoencoder architecture to train a model in near-real-time on a low-cost embedded microcontroller.

Related Work

A variety of methods to detect attacks/intrusions on the CAN bus using machine learning have been proposed.

In [1], a traditional neural net and multi-layer perceptron are compared for their performance in detecting CAN bus intrusions. They are trained and tested on a static dataset of CAN bus message traces. No effort is made to optimize the models for implementation on an embedded device, but the models achieve respectable performance.

Researchers in [2] made use of a GRU model to achieve high performance with a lightweight model intended to be deployed on embedded hardware. They also propose a system architecture for deploying, updating and training the model over-the-air (OTA) which

would be desired for real-world applications to keep model performance up to date as CAN bus attacks evolve.

[3] and [4] make use of support vector machines (SVMs) to do one-class classification which enables models to be trained only on 'good' CAN bus data and have the models infer when anomalous behavior is taking place without needing training data for those scenarios. This is appealing since attack methods are likely unknown at the time of training, and having one-class models would allow detection of novel attacks before they impact any vehicle.

[5] is the most recent paper I was able to find and proposes another model type, TPE-LightGBM algorithm. They achieve good performance and the model is lightweight, although they do not deploy it on embedded hardware.

Three papers are especially of interest for this project. [7] Outlines the use of autoencoders as One Class Classifiers (OCC), and the addition of a Batch-Wise Feature Weighting as a means of combating the autoencoders inherent skill at generalizing which reduces its accuracy in OCC applications. [11] Attempts to use an autoencoder for CAN bus intrusion detection, but with low accuracy and without investigating the impact of adding a BFW. In [10], researchers describe the methods used to accumulate the can-train-and-test dataset, which will be used in this project to train and test the model.

This project goes beyond previous work in a few ways. First, the work in [7] to define a BFW for use in autoencoders has not been applied to CAN bus intrusion detection. Further, although in [11], an autoencoder is used for CAN bus intrusion, and the model is run on embedded hardware, it is not run in real time, nor is an attempt made to train the model on the edge.

Project Deliverables

In this project, I will create a machine learning pipeline which uses an autoencoder to detect anomalies or intrusions of the CAN bus. I will compare the effects of using a Batch-Wise Feature Weighting as described in [7] and other tuning of hyperparameters to improve model recall and precision.

For this project I will develop the ML pipeline on my laptop, without needing to optimize for an embedded platform, before porting my architecture to the Arduino Uno Q, where I will replay the CAN message traces, confirming my model's performance in real-time detection. If time permits, I may also investigate training on the edge.

Project Gantt Chart

RTES 2025 GANTT CHART															
Date	9/25/2025		10/9/2025		10/23/2025		11/6/2025		11/20/2025		12/4/2025				
Week	5	6	7	8	9	10	11	12	13	14	15				
Ckpt	1		2		3		4		5		6				
Tasks															
Background Reading															
Selection of Hardware															
Selection of Machine Learning Strategy															
Selection of Dataset(s)															
Programming Non-Embedded ML Model															
Analysis of Non-Embedded ML Model															
Hardware System Design															
Hardware Assembly															
Port ML Model to Embedded Platform															
Analysis of Embedded ML Model															
Final Report															

Current progress report (Match):

- Completion of autoencoder_example (proof of concept for autoencoder with BFW using MNIST data set.
- Investigation and implementation of Batch-Wise Feature Weighting (BFW) as described in [7]
- Much progress on autoencoder for CAN bus intrusion detection.
    - Pipeline to download, extract features from and train on can-train-and-test data
    - parameterized autoencoder model using pytorch which can be configured with or without BFW
- instructions for reproducibility for all work to date

I accomplished a lot of what I set out to for this checkpoint, but I did underestimate the amount of work the getting this 'non-embedded' model up and running would be. I am currently using a rather naïve feature extraction on the can-train-and-test data, and I am getting reconstruction error that is too large even for the known class. I need to experiment with tuning the hyperparameters of my model, as well as investigate more feature extractions. Luckily I have built in another week for this work in my Gantt Chart, so I think I am still on track overall.

Over the next 2 weeks, as can be seen on the Gantt chart, my first priority will be to find parameters and feature extraction that get model performance into an acceptable range, and then to do a more formal analysis of this model which will include reproducibility instructions. Then, I will assemble my hardware layout. I have received all necessary components in the mail so I should be ready to go on this part.

I have moved my goal of being able to train the model on an embedded device into a 'reach' category, since I am not sure this will be possible on the hardware I have selected and I am not sure if I will have time to fully investigate this this semester.

Supporting Evidence (Factual): • https://github.com/finncg1234/RTES-2025

README: Has instructions for how to run all of the code that I added this checkpoint
https://github.com/finncg1234/RTES-2025/blob/main/README.md
autoencoder_example: this is a proof-of-concept for my autoencoder with BFW using the MNISt dataset. See the README above for detailed instructions about how to run this code and produce a pdf write-up of the results.
https://github.com/finncg1234/RTES-2025/tree/main/autoencoder_example

Check out this pdf which was produced by running the autoencoder_example project for 1000 epochs, which takes a while and I wouldn't expect a reviewer to do, especially if they don't have an Nvidia GPU:
https://github.com/finncg1234/RTES-2025/blob/main/autoencoder_example/autoencoder_example_1000_epochs.pdf

autoencoder_can: the main project files so far. This is a work in progress, but there are also instructions for how to run what I have so far if desired in the README
https://github.com/finncg1234/RTES-2025/tree/main/autoencoder_can

Skill Learning Report:

ML/AI: I have relatively little experience with machine learning, but for this checkpoint I implemented multiple versions of an autoencoder and assessed their performance based on metrics I saw used in other papers. I had to consider feature extraction and datasets.

Research: I have read 13 papers, doing my best to absorb the most relevant information to my project. I have honed in on machine learning models and their tradeoffs for real-time systems.

Mathematics: My investigations into ML architecture forced me to brush up on my linear algebra and multivariable calculus. I had to read and understand matrix equations and had to learn about gradients and Lagrange Optimization.

Technical Writing: This report is part of the work I did for this checkpoint.

Self-Evaluation:

Scope: 100

By Checkpoint 3, I have a really good idea of where this project is going. I have read a lot of papers to see what the state-of-the-art is and I am confident that my project goes beyond what is out there by combining some of the results of previous works and making a more concerted effort to get my model to run on hardware and in real-time than I have seen in other studies.

Match: 95

I was hoping to be a little bit farther in my development of my ML pipeline, but setbacks in tuning hyperparameters and settling on a good feature extraction have delayed me a bit. I am still on track with my plan at the beginning of the semester, but its taking longer than expected to get the model just right.

Factual: 100

I made a strong effort to make all of my findings reproducible. I am especially proud of the way I tied up my proof-of-concept mini-project with a automated report and analysis. Anyone with the required hardware (Nvidia GPU, probably) and software (python, pip, pytorch,…) can run my code and see the work I did.

I have similar instructions for the main part of my project, but as I state above, this is still a work in progress, so it is not really reproducible in the same way.

References:

[1] F. Amato, L. Coppolino, F. Mercaldo, F. Moscato, R. Nardone, and A. Santone, "CAN-Bus Attack Detection With Deep Learning," IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 8, pp. 5081–5090, Aug. 2021, doi: 10.1109/TITS.2020.3046974.

[2] 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, Article ID 5827056, 11 pages, 2022, doi: 10.1155/2022/5827056.

[3] J. Guidry, F. Sohrab, R. Gottumukkala, S. Katragadda, and M. Gabbouj, "One-Class Classification for Intrusion Detection on Vehicular Networks," in Proc. 2023 IEEE Symposium Series on Computational Intelligence (SSCI), Mexico City, Mexico, 2023, pp. 1176–1182, doi: 10.1109/SSCI52147.2023.10371899.

[4] C. Chupong, N. Junhuathon, K. Kitwattana, T. Muankhaw, N. Ha-Upala, and M. Nawong, "Intrusion Detection in CAN Bus using the Entropy of Data and One-class Classification," in Proc. 2024 Int. Conf. on Power, Energy and Innovations (ICPEI), Nakhon Ratchasima, Thailand, 2024, pp. 157–160, doi: 10.1109/ICPEI61831.2024.10748816.

[5] L. Liang et al., "Intrusion Detection Model for In-vehicle CAN Bus Based on TPE-LightGBM Algorithm," in Proc. 2025 IEEE 34th Wireless and Optical Communications Conference (WOCC), Taipa, Macao, 2025, pp. 419–423, doi: 10.1109/WOCC63563.2025.11082193.

[6] J. N. Brewer and G. Dimitoglou, "Evaluation of Attack Vectors and Risks in Automobiles and Road Infrastructure," in Proc. 2019 Int. Conf. on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019, pp. 84–89, doi: 10.1109/CSCI49370.2019.00021.

[7] Y. Liao and B. Yang, "To Generalize or Not to Generalize: Towards Autoencoders in One-Class Classification," in Proc. 2022 Int. Joint Conf. on Neural Networks (IJCNN), Padua, Italy, 2022, pp. 1–8, doi: 10.1109/IJCNN55064.2022.9892812.

[8] J. Lee, S. Park, S. Shin, H. Im, J. Lee, and S. Lee, "ASIC Design for Real-Time CAN-Bus Intrusion Detection and Prevention System Using Random Forest," IEEE Access, vol. 13, pp. 129856–129869, 2025, doi: 10.1109/ACCESS.2025.3585956.

[9] Z. Bi, G. Xu, G. Xu, M. Tian, R. Jiang, and S. Zhang, "Intrusion Detection Method for In-Vehicle CAN Bus Based on Message and Time Transfer Matrix," Security and Communication Networks, vol. 2022, Article ID 2554280, 19 pages, 2022, doi: 10.1155/2022/2554280.

[10] B. Lampe and W. Meng, "can-train-and-test: A New CAN Intrusion Detection Dataset," in Proc. 2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall), Hong Kong, 2023, pp. 1–7, doi: 10.1109/VTC2023-Fall60731.2023.10333756.

[11] B. M. Tóth and A. Bánáti, "Autoencoder Based CAN BUS IDS System Architecture and Performance Evaluation," in Proc. 2025 IEEE 19th Int. Symp. on Applied Computational Intelligence and Informatics (SACI), Timisoara, Romania, 2025, pp. 000099–000104, doi: 10.1109/SACI66288.2025.11030168.

[12] M. Kemmler, E. Rodner, E.-S. Wacker, and J. Denzler, "One-class classification with Gaussian processes," Pattern Recognition, vol. 46, no. 12, pp. 3507–3518, 2013, doi: 10.1016/j.patcog.2013.06.005.

[13] F. Sohrab, J. Raitoharju, M. Gabbouj, and A. Iosifidis, "Subspace Support Vector Data Description," in Proc. 2018 24th Int. Conf. on Pattern Recognition (ICPR), Beijing, China, 2018, pp. 722–727, doi: 10.1109/ICPR.2018.8545819.