CAN Bus Intrusion Detection based on Auxiliary Classifier GAN and Out-of-Distribution Detection

QINGLING ZHAO, Nanjing University of Science and Technology, China
MINGQIANG CHEN, Nanjing University of Science and Technology, China
ZONGHUA GU, Umeå University, Sweden
SIYU LUAN, Umeå University, Sweden
HAIBO ZENG, Virginia Tech, USA
SAMARJIT CHAKRABORY, The University of North Carolina at Chapel Hill, USA

With the rapid advances in Connected and Automated Vehicles, the issue of automotive cyber-security is gaining increasing attention. The Controller Area Network (CAN) is a ubiquitous bus protocol present in the Electrical/Electronic (E/E) systems of almost all vehicles. It is vulnerable to a range of attacks once the attacker gains access to the bus through the vehicle's attack surface. We address the problem of Intrusion Detection on the CAN bus, and present a series of methods based on two classifiers trained with Auxiliary Classifier Generative Adversarial Network (ACGAN) to detect and assign fine-grained labels to Known Attacks, and also detect the Unknown Attack class in a dataset containing a mixture of (Normal + Known Attacks + Unknown Attack) messages. The most effective method is a cascaded two-stage classification architecture, with the multi-class Auxiliary Classifier in the first stage for classification of Normal and Known Attacks, passing Out-of-Distribution (OOD) samples to the binary Real-Fake Classifier in the second stage for detection of the Unknown Attack class. Performance evaluation demonstrate that our method achieves both high classification accuracy and low runtime overhead, making it suitable for deployment in the resource-constrained in-vehicle environment.

CCS Concepts: • **Security and privacy** → **Intrusion detection systems**.

Additional Key Words and Phrases: Automotive Security, Controller Area Network, intrusion detection, Deep Learning, GAN

ACM Reference Format:
Qingling Zhao, Mingqiang Chen, Zonghua Gu, Siyu Luan, Haibo Zeng, and Samarjit Chakrabory. 2021. CAN Bus Intrusion Detection based on Auxiliary Classifier GAN and Out-of-Distribution Detection. *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2021), 31 pages. https://doi.org/10.1145/3506703

Q. Zhao and M. Chen's work was supported by NSFC Grant No. 61902185 and Jiangsu Provincial NSF Grant No. BK20190448. S. Chakraborty's work was supported by NSF Grant No. 20389600.

Q. Zhao and M. Chen, the PCA Lab, Key Lab of Intelligent Perception and Systems for High-Dimensional Information of Ministry of Education, and Jiangsu Key Lab of Image and Video Understanding for Social Security, School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, Jiangsu 210094, China, email: {ada_zhao, chenmingqiang}@njust.edu.cn; Z. Gu, Department of Applied Physics and Electronics, Umeå University, 90187 Umeå, Sweden; email: zonghua.gu@umu.se; H. Zeng, Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061, USA; email: hbzeng@vt.edu; S. Chakraborty, Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599, USA; email: samarjit@cs.unc.edu.

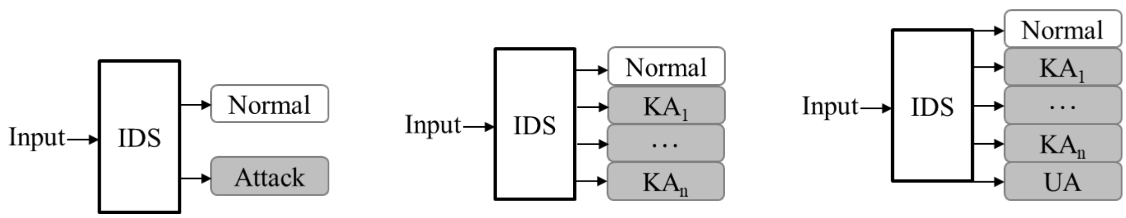
1 INTRODUCTION

Modern Connected and Automated Vehicles (CAVs) are prototypical Cyber-Physical Systems [10], where the in-vehicle Electrical/Electronic (E/E) system interacts closely with its physical surroundings to form a sense-compute-actuate feedback loop, including the vehicle itself and the external driving environment. A CAV has a large attack surface. The attacker may compromise its cyber-security by gaining access to its internal networks through the vehicle's attack surface, either physically through the On-Board Diagnostics (OBD-II) connector, or wirelessly through the interface of the On-Board Unit (OBU) (e.g., cellular, WiFi, Bluetooth). The attacker may target non-essential functions, e.g., window lifters, indicator lights, the signal horn or the display panel, to cause minor annoyances, or safety-critical functions, e.g., acceleration, brake, steering control inputs, to cause more severe consequences.

The automotive E/E system is a distributed system consisting of multiple Electronic Control Units (ECUs) interconnected with different in-vehicle networking protocols, including Ethernet, FlexRay, MOST (Media Oriented Systems Transport), CAN, and LIN (Local Interconnect Network). The CAN bus is used for low or medium-bandwidth in-vehicle networking to connect the E/E components, including ECUs, sensors, and actuators. We focus on the CAN bus in this paper, which is ubiquitously deployed in almost all vehicles, regardless of the price range or the manufacturer. The CAN bus is vulnerable to attacks due to several factors: broadcast transmission, where all nodes can hear all message transmissions on the bus; ID-based priority arbitration; and lack of authentication/encryption mechanisms. One possible defense mechanism is Message Authentication (MA) for defending against message spoofing attacks, where the sender adds Message Authentication Codes (MAC) to messages, and the receiver verifies the MAC, based on Public Key Cryptography. This approach is constrained by the limited computing power of ECUs and the small payload size of CAN messages (8 Bytes). Proposed solutions include: adding hardware coprocessors to offload computation-intensive cryptographic algorithms from the ECU [31]; selectively adding coprocessors to a partial set of ECUs to meet timing constraints while minimizing cost [21]; truncating the MAC to reduce its length, and performing successful authentication of several consecutively received messages [21, 47]; dynamically adjusting the MAC size to maximize it while meeting timing constraints [54]; security-aware obfuscated priority assignment for CAN with Flexible Data-rate (CAN-FD) [53], and others. Despite these research advances, MA on the CAN bus is not yet widely adopted in the automotive industry due to its high cost or high complexity. In addition, MA can only defend against message spoofing attacks, not other types of attacks such as DoS or Fuzzing.

A more practical alternative approach is to adopt an onboard *Intrusion Detection System (IDS)* to monitor the CAN bus traffic to detect possible attacks. Since CAN is a multi-master broadcast bus protocol, the IDS may be deployed as a separate dedicated node on the CAN bus, implemented as either hardware (FPGA or ASIC) or integrated into one of the ECUs as a software application. Upon detecting intrusions/attacks, the Intrusion Prevention System (IPS) may take immediate countermeasures, e.g., sending error frames to stop the current transmission [18]. In addition, a detailed log of the onboard security events may be sent to the Security Operations Center in the cloud, which performs in-depth post-attack analysis of the collected data both for single vehicles and the whole fleet, e.g., root cause analysis, impact analysis, etc., for developing threat response strategies. The automotive software standard AUTOSAR (AUTomotive Open System ARchitecture) is in the process of introducing a new standard for onboard IDS [15].

There has been significant related work on CAN bus IDS (as discussed in Section 4). When formulated as a machine learning problem, existing methods either perform coarse-grained binary classification of Normal vs. Attack (Fig. 1(a)); or perform multi-class classification with $n + 1$ classes, including Normal and *n Known Attacks (KAs)*, i.e., to detect KAs and assign fine-grained labels to them, but cannot detect any *Unknown Attack (UA)* (Fig. 1(b)). In this paper, we consider the novel problem formulation of multi-class classification with $n + 2$ classes, including Normal, n KA classes and one UA class (Fig. 1(c)), i.e., to detect both KAs and any UA, and also assign fine-grained labels to detected KAs. To justify our problem formulation: on the one hand, it is important to be able to assign fine-grained labels to KAs, since knowing the specific attack type can be quite useful for selecting appropriate counter-measures and performing post-attack analysis; on the other hand, it is important to be able to detect any UA, since attackers may come up with novel zero-day attacks that may not fit the patterns of existing attacks.



(a) Binary classification of Normal vs. Attack

(b) Multi-class classification of Normal and n KAs

(c) Multi-class classification of Normal, n KA classes and one UA class (addressed in this paper).

Fig. 1. Machine Learning-based approaches to CAN bus IDS.

We propose and evaluate four different methods[1], and demonstrate that the best-performing method is the one based on Auxiliary Classifier Generative Adversarial Network (ACGAN) [38] in combination with Out-of-Distribution (OOD) detection [24], where the Auxiliary Classifier (AC) in ACGAN is used for assigning fine-grained labels to KAs, OOD detection is used for preliminary detection of the UA class, and the Real-Fake Classifier (RFC) in ACGAN is used for final determination of the UA class.

Table 1 contains frequently-used abbreviations used in this paper.

Table 1. Abbreviations used in this paper.

IDS	Intrusion Detection System
KA	Known Attack
UA	Unknown Attack
FFNN	Feed-Forward Neural Network
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
D	GAN Discriminator
G	GAN Generator
OOD	Out-of-Distribution
MSP	Maximum SoftMax Probability

The rest of the paper is structured as follows: we present the problem formulation and our approach in Section 2, including the input encoding method, ACGAN, OOD detection, and our four proposed methods; performance evaluation in Section 3; discussions of related work in Section 4; and conclusions and future work in Section 6.

[1]Our code is open-source, available at https://github.com/leyiweb/CAN_GAN_Anomaly

2 PROBLEM FORMULATION AND OUR APPROACH

2.1 Introduction to the CAN Bus Protocol

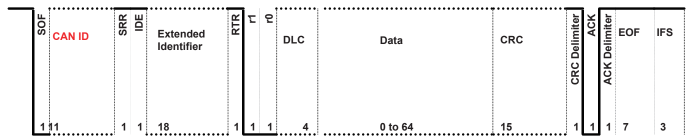
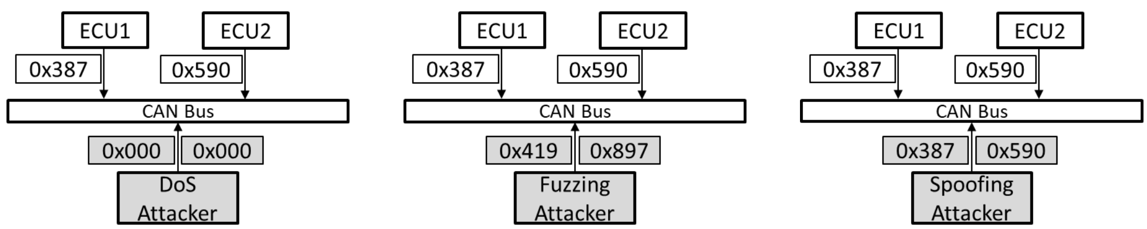


Fig. 2. Format of a standard CAN message.

Fig. 2 shows the format of a CAN message, which consists of a header (CAN ID) with the standard format of 11 bits and a payload of up to 8 Bytes. (We assume the standard CAN ID length of 11 bits in this paper, but the extended CAN ID length of 29 bits can be easily handled by adjusting the input encoding.) Each bit in the CAN ID is either dominant (0) or recessive (1), and the dominant bit wins in case of a collision on the bus. This implies that the CAN ID determines message priority, and a message with a smaller CAN ID has higher priority. The broadcast nature of the CAN bus creates opportunities for attackers to inject malicious messages from a compromised ECU, and the simple priority arbitration protocol means that malicious messages with smaller CAN IDs can cause undue delays to legitimate messages.



(a) Dos Attack (b) Fuzzing Attack (c) Spoofing Attack

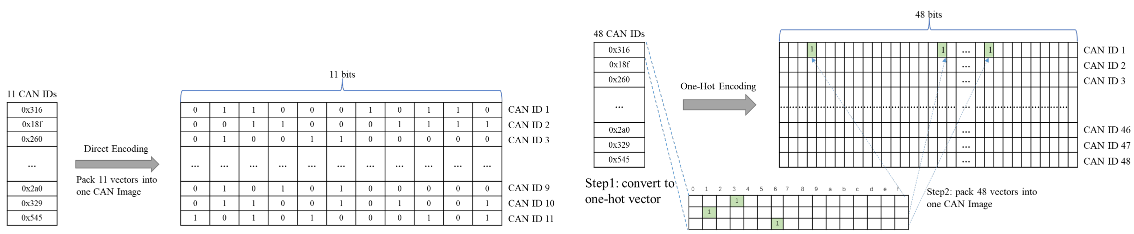
Fig. 3. Three common types of attacks on the CAN bus.

Fig. 3 illustrates the three common types of attacks on the CAN bus, addressed in this paper:

- **DoS** (Denial-of-Service): the attacker injects high-frequency messages with high-priority (e.g., with CAN ID 0x000 or some other ID that is known to be smaller than all legitimate message IDs) to flood the bus and occupy the bus bandwidth, in order to prevent the legitimate messages from transmitting successfully, e.g., in Fig. 3(a), the messages with CAN IDs 0x387 and 0x590 are both delayed by the attack messages with CAN ID 0x000.
- **Fuzzing**: the attacker injects CAN messages with random CAN ID, to cause delays to certain legitimate messages with lower priority (larger CAN IDs), e.g., in Fig. 3(b), the message with CAN ID 0x387 sent by the legitimate node ECU1 is not delayed by the two attack messages due to its smaller ID value; the message with CAN ID 0x590 sent by the legitimate node ECU2 is delayed by the attack message with CAN ID 0x419, but not by the attack message with CAN ID 0x897. This type of attack is more stealthy than DoS attack thanks to the randomness of the CAN IDs of the injected messages.

- **Spoofing**: the attacker injects messages with specific CAN IDs that belong to legitimate nodes, which may be captured by observing the traffic on the CAN bus before the attack, and with modified payloads, e.g., the two attack messages have the same CAN IDs 0x387 and 0x590 as the legitimate nodes. The attacker's goal is not to cause delays to normal messages, but to inject false signals to be read by other nodes. (In the public car-hacking dataset [50], two types of spoofing attacks are included, including **GEAR** for Gear-Shift and **RPM** for the engine's rotation speed measured with Revolutions-Per-Minute.)

2.2 Input Encoding



(a) Direct encoding (b) One-hot encoding

Fig. 4. Two different ways of encoding a sequence of CAN IDs into a 2D image.

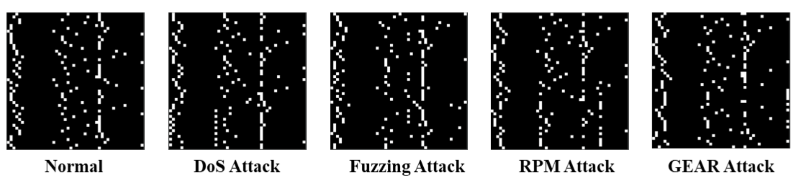


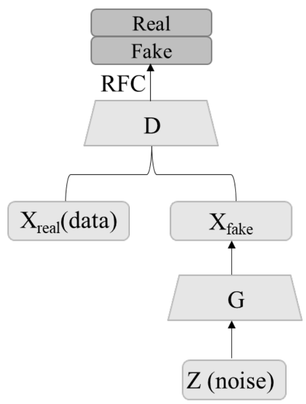
Fig. 5. Example 48×48 CAN images for normal data and different types of KAs.

In order to apply Deep Learning to CAN bus intrusion detection, we need to encode a sequence of CAN IDs into a format suitable for input to a Feed-Forward Neural Network (FFNN) (including Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN)), which takes an image as input data, which may be a 1D time series or a 2D image of pixels. We perform a preprocessing step to encode a consecutive sequence of CAN IDs into a 2D Image. Fig. 4(a) shows one method of constructing an $N \times 11$ 2D square image by stacking a set of N (here $N = 11$) CAN IDs, each 11-bit long [45]. We adopt another method of constructing a $N \times 48$ 2D square image with one-hot encoding [42], shown Fig. 4(b). Each 11-bit CAN ID is converted into a 48-bit vector, which consists of three 16-bit vectors, each one-hot encoding one hex digit, e.g., the ID of $0x316$ in hexadecimal format is converted into three 16-bit vectors, each encoding a hex number 3, 1, 6, respectively. We then stack a set of N (here $N = 48$) CAN IDs, each with length 48 bits, to form a 48×48 2D square image. The term "input sample" or "sample" are used to refer to each CAN image. This approach to CAN image construction is inspired by GIDS (GAN based Intrusion Detection) [42].

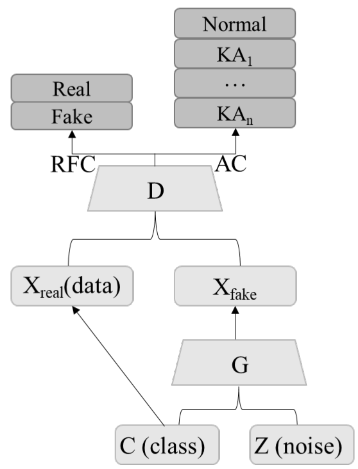
- We choose the one-hot encoding method, which was shown [42] to result in larger differences and easier separation between normal images and attack images than the direct encoding method.

- The input size N, i.e., the number of CAN messages grouped into a single CAN image, is an important hyper-parameter that can be selected to tradeoff between detection accuracy and computation overhead. Experimental results in [42] indicate that the input size of 64 achieves the best accuracy. (In our experiments, we set $N = 48$ to form a square image, which is the most common image shape in computer vision.)

Fig. 5 shows some example CAN images obtained with one-hot encoding (refer to Section 3.1 for details on the different attack types.) The differences among them may not be apparent by visual inspection, as the number of attack messages may be a small percentage of the total 48 messages. However, these differences enable the application of powerful Machine Learning (ML), esp. Deep Learning (DL) methods, to distinguish between them.



(a) GAN



(b) ACGAN

Fig. 6. Regular GAN vs. ACGAN.

2.3 GAN and ACGAN

Fig. 6(a) shows Generative Adversarial Network (GAN) [19], a framework for joint training of two models simultaneously, generator G and discriminator D. G aims to capture the data distribution. It takes as input a random noise vector z from an arbitrary latent distribution (noise prior), and generates a synthetic (fake) sample $X_{fake} = G(z)$. D aims to distinguish between real samples and fake samples generated by G. It receives as input sample, either a real sample X_{real} or a fake sample X_{fake} from G, and outputs a probability distribution $P(S \mid X)$ over two possible sample sources S={real, fake}. D is a CNN backbone for feature extraction, with a binary classifier head, called the *Real-Fake Classifier (RFC)* in this paper. The RFC is typically implemented as a Sigmoid function with optional fully-connected layers preceding it. (A minor technical detail is whether the RFC is viewed as part of D or separate from D, which consists of the CNN backbone only. We take the latter view in this paper.) The GAN loss function is defined as the loglikelihood of the correct source:

$$L = E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})] \tag{1}$$

D is trained to maximize L, i.e., to use the RFC to distinguish between real vs. fake samples, and G is trained to minimize the second term in L, i.e., to generate realistic fake samples that fool the RFC of D into classifying it as real.

Fig. 6(b) shows ACGAN [38], an extension of GAN with an additional multiclass *Auxiliary Classifier (AC)* head, a SoftMax layer with optional fully-connected layers preceding it, which shares the same CNN backbone D with the RFC for feature extraction. Every generated sample has a corresponding class label $c \sim p_c$ in addition to the noise z. G uses both to generate class-conditional fake samples $X_{fake} = G(c, z)$. For each input sample, D performs two tasks with its two classifier heads: the binary RFC classifies it into either real or fake by outputting a probability distribution $P(S \mid X)$, regardless of its class label c; the multiclass AC assigns it a class label by outputting a probability distribution $P(C \mid X)$ over all possible class labels, regardless of its source S being real or fake. The loss function for ACGAN has two parts:

$$
\begin{aligned}
L_S &= E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})] \\
L_C &= E[\log P(C = c \mid X_{real})] + E[\log P(C = c \mid X_{fake})]
\end{aligned} \tag{2}
$$

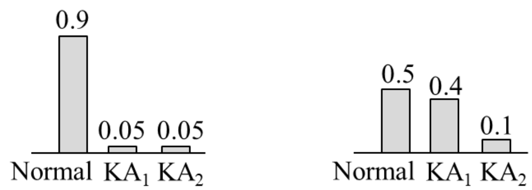
D is trained to maximize $L_C + L_S$, and G is trained to maximize $L_C - L_S$. In addition to L_S, the same loss function of GAN (Eq. (1)), the additional term L_C in ACGAN's loss function forces G to generate realistic fake samples with some given class label c, and the AC of D to perform accurate classification of both real and fake samples with label c. It has been shown that this small extension to the standard GAN helps stabilize training and learn better representations [38].

The original design intention of ACGAN is to use the AC to help improve the GAN training process, and train a better G that can generate diverse and high-resolution class-conditional photo-realistic image samples; D serves in an auxiliary role to help the training of G, and is not useful after training. In contrast, we apply ACGAN in an unconventional way: our goal is not to generate photo-realistic image samples, but to train a better D, with its two classifier heads RFC and AC; G serves in an auxiliary role to help the training of D, and is not useful after training.

2.4 Out-of-Distribution (OOD) Detection

OOD samples refer to samples that fall outside of the distribution of the training dataset, i.e., outliers/anomalies. The opposite of OOD is In-Distribution (InD), or inliers/normal data. An OOD detector [24] is a binary classifier that predicts an InD or OOD label to an input sample. OOD detection is an active research area with a wide range of algorithms and techniques [9], including our previous work [35] on using Isolation Forest (IF) or Local Outlier Factor (LOF) for outlier detection in one or more hidden layers of a CNN.

Hendrycks and Gimpel [24] presented a simple baseline method of using the predicted *Maximum SoftMax Probability (MSP)* to separate OOD samples (with low MSP) from InD samples (with high MSP) based on a threshold th, called the *OOD threshold* in this paper. An input sample is determined to be OOD if its MSP does not exceed the OOD threshold. As an example, Fig. 7 shows two possible probability distributions as output from the SoftMax layer for two different input samples. In both cases, the classifier predicts the MSP class Normal, but the classifier has different levels of confidence in its prediction. Suppose we set the OOD threshold $th = 0.8$, then the input sample X_1 corresponding to the distribution $P(C \mid X_1)$ in Fig. 7(a) has $MSP = 0.9 \geq th$ and is determined to be Normal; the input sample X_2 corresponding to the distribution $P(C \mid X_2)$ in Fig. 7(b) has $MSP = 0.5 < th$ and is determined to be OOD. The MSP method is not the most accurate OOD detection algorithm, since Neural Networks often give incorrect yet over-confident predictions, but it is widely used in practice, since its accuracy is often good enough, and it is very efficient, incurring no additional overhead beyond the forward inference time. Most of the more sophisticated OOD detection algorithms [9], including our own work based on Isolation Forest (IF) and Local Outlier Factor (LOF) [35], have too high runtime overhead to be used for CAN bus IDS.



(a) Probability distribution $P(C \mid X_1)$ for input sample X_1, with $MSP = 0.9$.

(b) Probability distribution $P(C \mid X_2)$ for input sample X_2, with $MSP = 0.5$.

Fig. 7. Two different SoftMax distributions for illustration of OOD detection based on MSP.

2.5 Four Methods for CAN bus IDS

2.5.1 General Observations and Statements. The CAN bus IDS may be developed by a supplier as a general-purpose product sold to OEMs (Original Equipment Manufacturers), hence it must be able to handle a wide range of diverse applications across different OEMs and vehicle models. Hence the Normal class is likely to contain CAN messages from different applications with diverse features, i.e., an irregular, multimodal distribution in the feature space. In contrast, each KA is likely to have its own distinct features, with a concentrated, sharp distribution in the feature space. The UA class by definition will have diverse features, since it is novel and unseen during training time. If there is no UA, i.e., all attacks are KAs with available training data, then it is straightforward to train a classifier with Supervised Learning for either binary classification of Normal vs. Attack, or multiclass classification of Normal and n KAs. However, the problem becomes more difficult if we want to detect both KAs and any UA, and also assign fine-grained labels to detected KAs.

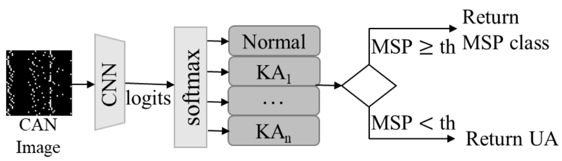
Table 2. Important statements as design rationales.

S1	A UA sample is more likely to be OOD (with MSP < th) than a Normal or a KA sample.
S2	A Normal sample is more likely to be OOD (with MSP < th) than a KA sample.
S3	A UA sample is more likely to be misclassified as Normal than as one of the KAs.
S4	A UA sample is more likely to be misclassified as Normal than a KA sample being misclassified as Normal.

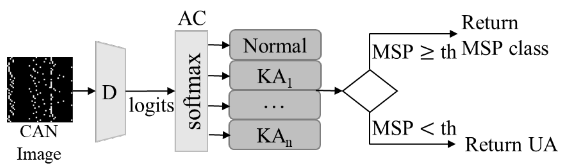
We make a set of important statements in Table 2. The reason for S1 is that UA samples are not in the training dataset. Hence they are likely to result in lower MSP values. The reason for S2 is that Normal samples may have more diverse features than any of the KAs, hence they are more difficult to classify and result in lower MSP than any of the KA classes. The reasons for S3 and S4 are that Normal samples and UA samples are more easily confused with each other, whereas each KA's samples are more easily classified and distinguished from Normal samples and UA samples. These statements will be confirmed by experimental results later.

We propose four CAN bus IDS methods as shown in Fig. 8. Assuming n KAs, the fine-grained known/unknown IDS performs multiclass classification among $n+2$ classes $C = \{\text{Normal}, \text{KA}_1, \ldots, \text{KA}_n, \text{UA}\}$, including $n+1$ classes (Normal and n KAs) with labeled samples in the training dataset, plus one additional UA class not in the training dataset. The four methods share the common first step of performing multiclass classification among $n+1$ classes $C = \{\text{Normal}, \text{KA}_1, \ldots, \text{KA}_n\}$, including Normal and n KAs, This $(n+1)$-class classifier may be trained with Supervised Learning, as in CNN-Th, or may be trained as the AC in ACGAN. They differ in the next step of identifying/classifying

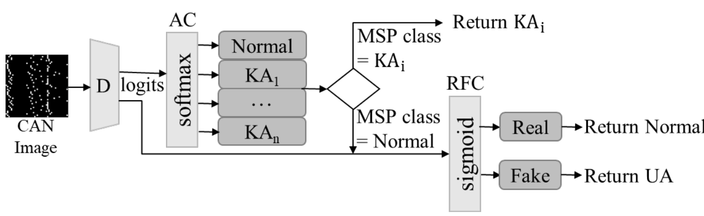
UA samples, based on the statements in Table 2. The RFC in ACGAN is used by two of the four methods, which is a binary classifier with label real corresponding to one of the $n + 1$ classes (Normal and n KAs), and the label fake corresponding to the UA class. (Note that we assign the class label UA to all possible UAs, since we do not distinguish among different types of UAs.)



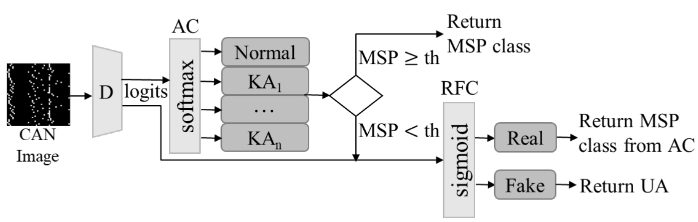
(a) CNN-Th



(b) ACGAN-AC-Th



(c) ACGAN-AC-RFC-Normal



(d) ACGAN-AC-RFC-Th

Fig. 8. Four proposed methods for CAN bus IDS.

2.5.2 CNN-Th. Fig. 8(a) shows the CNN-Th method. The CNN is used for assigning fine-grained labels to KAs, and OOD detection is used for detection of the UA class (with MSP < th). Its design rationale is Statement S1 in Table 2: "A UA sample is more likely to be OOD than a Normal or a KA sample.". CNN-Th adopts the simplistic assumption that all OOD samples should be labeled as UA. As shown in Equation (3), we use the MSP method [24] for OOD detection: 1) For a given input sample X, compute the MSP $P(c^* \mid X)$ over the known $n+1$ classes $c_i \in C = \{\text{Normal}, \text{KA}_1, \ldots, \text{KA}_n\}$;

Table 3. With ACGAN-AC-RFC-Normal, samples with MSP class Normal are passed to the RFC.

InD/OOD \ Predicted	Normal	KA_i
InD (MSP ≥ th)	Pass to RFC	return KA_i
OOD (MSP < th)		return KA_i

Table 4. With ACGAN-AC-RFC-Th, OOD samples (with MSP < th) are passed to the RFC.

InD/OOD \ Predicted	Normal	KA_i
InD (MSP ≥ th)	return Normal	return KA_i
OOD (MSP < th)	Pass to RFC	

2) if the MSP exceeds the OOD threshold ($MSP \geq th$), we trust this prediction and return the MSP class c^*; otherwise, the input sample is determined to be OOD and labeled as UA.

$$c^* = \begin{cases} \text{argmax}_{c_i \in C} P(c_i \mid X) & \text{if } P(c^* \mid X) \geq th \\ UA & \text{Otherwise} \end{cases} \quad (3)$$

2.5.3 ACGAN-AC-Th. Fig. 8(b) shows the ACGAN-AC-Th method. It is similar to CNN-Th except that the CNN is replaced by the AC in ACGAN, which is used for assigning fine-grained labels to KAs; OOD detection is used for detection of the UA class (with MSP < th). The RFC in ACGAN is not used here. Our experimental results indicate that the AC trained with ACGAN generally has better performance than a CNN with similar architecture but trained with Supervised Learning, since the training dataset for the AC consists of both the labeled training dataset (the real samples) and the synthetic (fake) samples generated by G. If G is well-trained and generates realistic fake samples, then this is an effective form of data augmentation, hence the training dataset for the AC is enlarged significantly compared to the CNN, which only uses the real samples. We expect the ACGAN-based approach to be especially beneficial in the case of unbalanced datasets with limited attack data samples, which is a common situation in practice.

2.5.4 ACGAN-AC-RFC-Normal. Fig. 8(c) shows the ACGAN-AC-RFC-Normal method, a cascaded two-stage classification architecture. The AC in ACGAN is used for assigning fine-grained labels to KAs; assuming the KAs are classified correctly but UA samples may be misclassified as the Normal class, the RFC in ACGAN is used for determination of the UA class. Its design rationale includes statements S3 and S4 in Table 2: "A UA sample is more likely to be misclassified as Normal than as one of the KAs" and "A UA sample is more likely to be misclassified as Normal than a KA sample being misclassified as Normal". The logits (feature vector from the last hidden layer of the CNN backbone D) are passed to both the first-stage multiclass AC, and the second-stage binary RFC. If the AC predicts one of the KAs as the MSP class among the $n + 1$ classes, we trust its prediction and return that class label. Otherwise, the AC predicts Normal as the MSP class. We do not trust its prediction, and refer to the second-stage RFC. If the RFC predicts real, then return Normal; if it predicts fake, then return UA.

2.5.5 ACGAN-AC-RFC-Th. Fig. 8(d) shows the ACGAN-AC-RFC-Th method, another cascaded two-stage classification architecture. The AC in ACGAN is used for assigning fine-grained labels to KAs, OOD detection is used for preliminary detection of the UA class (with MSP < th), and the RFC in ACGAN is used for final determination of the UA class. Its design rationale includes statements S1 and S2 in Table 2: "A UA sample is more likely to be OOD than a Normal or a KA sample" and "A Normal sample is more likely to be OOD than a KA sample" . Since not all OOD samples are UA samples, i.e., some OOD samples may be Normal or KA samples, we add the RFC to classify the UA samples more precisely. After the first-stage AC has computed the probability distribution $P(C \mid X)$ over the $n+1$ classes C for input sample X, if $MSP \geq th$, we trust its prediction and return that class label. Otherwise, the input sample X is determined to be OOD, so we do not trust the AC's prediction of the MSP class, and look at the second stage RFC. If the RFC predicts real, then return the MSP class label from the AC, which may be Normal or one of the n KAs; if RFC predicts fake, then return UA. (Note that the RFC has the same decision logic in ACGAN-AC-RFC-Normal: if the RFC predicts real, then return Normal, which is also the MSP class label from the AC.)

For further clarification, Tables 3 and 4 compare the decision logics of ACGAN-AC-RFC-Normal in Fig. 8(c) and ACGAN-AC-RFC-Th in Fig. 8(d) in tabular form.

3 PERFORMANCE EVALUATION

In this section, we present the details of constructing the training and test datasets in Section 3.1; the ACGAN model details in Section 3.2; performance comparisons among our four proposed methods in Section 3.3; performance comparisons with related work in Section 3.4, and timing performance in Section 3.5.

3.1 Dataset Construction

We use the public car-hacking dataset [50] in our experiments, which was constructed by the authors of [42] by logging CAN traffic via the OBD-II port from a real vehicle while message injection attacks were carried out. It consists of CAN message sequences for the Normal class and four attack classes: DoS, Fuzzing, Spoofing of GEAR or RPM messages (denoted as GEAR and RPM in short), as shown in Fig. 3.

We group a consecutive sequence of $N = 48$ CAN IDs into a CAN image with size 48×48, as shown in Fig. 4(b). Each normal CAN image contains 48 normal CAN messages, and each KA or UA CAN image contains 48 CAN messages with at least one attack message of the specific attack type (this corresponds to attack threshold of 1 in GIDS [42]). Inspired by the common method of emulating OOD samples in research works on OOD detection [9, 35], we emulate UA samples by samples of one KA that are excluded from the training dataset. Suppose we have 3 KAs (Fuzzing, RPM, GEAR) and 1 UA (DoS). The training dataset consists of samples of $n + 1 = 4$ classes, including Normal and $n = 3$ KAs (Fuzzing, RPM, GEAR). We exclude samples of the DoS attack class from the training dataset, in order to use them as UA samples during testing. The test dataset contains samples of $n + 2 = 5$ classes, including Normal, 3 KAs (Fuzzing, RPM, GEAR), and 1 UA (DoS). In our experiments, the training dataset consists of 40,000 samples (1.92M messages), 10,000 for each of the 4 classes (Normal and 3 KAs); the test dataset consists of a total of 25,000 samples (1.2M messages), 5,000 for each of the 5 classes (Normal, 3 KAs, and 1 UA). (In addition to using DoS as the UA, we also tried other KA/UA splits and obtained similar results.)

3.2 The ACGAN Model and Training Procedure

Table 5 shows the ACGAN model architecture based on the open-source repository [32], adapted to fit our problem setting, i.e., the set of classes for the AC. The CNN architecture used in the CNN-th method is identical to the Discriminator D, hence not included in the table. (Note that more sophisticated CNN architectures may be adopted to achieve even better performance, e.g., the Inception-ResNet [23] as used by Song et al. [45], but we view this as an orthogonal issue to our main contribution, which is the high-level architecture design of the two-stage classifier shown in Fig. 8.)

Table 5. Architecture and hyperparameter settings of the ACGAN. Each convolutional filter has size K × K, with stepsize S and padding P. BN denotes Batch Normalization.

Operation	Filter (K × K/S, P)	Output Feature Map	BN	Dropout	Activation
Generator					
Linear Embedding		$6 \times 6 \times 128$			
Upsample		$12 \times 12 \times 128$			
Convolution	$3 \times 3/1, 1$	$12 \times 12 \times 64$	✓		Leaky ReLU
Upsample		$24 \times 24 \times 64$			
Convolution	$3 \times 3/1, 1$	$24 \times 24 \times 32$	✓		Leaky ReLU
Upsample		$48 \times 48 \times 32$			
Convolution	$3 \times 3/1, 1$	$48 \times 48 \times 16$	✓		Leaky ReLU
Convolution	$3 \times 3/1, 1$	$48 \times 48 \times 1$			Tanh
Latent Dimension (z)	256				
Leaky ReLU slope	0.2				
BN momentum	0.8				
Optimizer	Adam(lr=0.0002, betas=(0.5,0.999))				
Discriminator					
Convolution	$3 \times 3/2, 1$	$24 \times 24 \times 16$		✓	Leaky ReLU
Convolution	$3 \times 3/2, 1$	$12 \times 12 \times 32$	✓	✓	Leaky ReLU
Convolution	$3 \times 3/2, 1$	$6 \times 6 \times 64$	✓	✓	Leaky ReLU
Convolution	$3 \times 3/2, 1$	$3 \times 3 \times 128$	✓	✓	Leaky ReLU
Linear (RFC)		2			sigmoid
Linear (AC)		4			logSoftmax
Leaky ReLU slope	0.2				
BN momentum	0.8				
Dropout	0.5				
Optimizer	SGD(lr=0.001 down to 0.0002, momentum=0.9)				

Our hardware platform for model training is a Linux workstation with CPU: Intel(R) Xeon(R) E5-2650 v4 @ 2.20GHz; RAM: 16GB; GPU: NVIDIA GeForce GTX 2080Ti. We use the deep learning framework PyTorch. We adopted the hyperparameter settings for GAN training primarily from [32], but adjusted the learning rates based on [29] to mitigate instability during training. For GAN training, we use the Adam optimizer for G with a constant learning rate of 0.0002; the SGD optimizer for D with an initial high learning rate of 0.001, gradually reduced to 0.0002 during the training process. It is well-known that model selection for GAN is challenging due to possible instability during training, and a longer training time does not necessarily lead to better performance [29]. Typically, the goal of GAN training is to train the G to generate realistic images, so one may select the G by visual inspection of generated images from different model checkpoints during the training process. However, our goal is different from the conventional GAN training, as we want to train the D, including both AC and RFC, to have good performance. Therefore, we adopt the approach from [29]: set aside 10% of the entire training dataset as the validation dataset, and use the remaining 90% as the actual training dataset. (Note that the typical k-fold cross-validation in Supervised Learning is not suitable for GAN training). The training process consists of 1000 epochs, with Batch Size 64. After each epoch, we checkpoint the model parameters, and measure the overall classification performance on the validation dataset in terms of the macro F1-score (computed with (6)). At the end of the training process, we select the D (the AC and the RFC), that achieves the highest macro

F1-score. This implies that we may select different D for different methods in Fig. 8, or even different OOD thresholds with the same method. In order to have fair performance comparisons, the hyperparameters for training the CNN are set to be the same as those for training D, in terms of the SGD optimizer and learning rate schedule. The entire training process lasts on average 11.5 hours for ACGAN, and 9 hours for CNN, including both training time on the GPU, and testing time on the validation dataset on the CPU.

3.3 Performance Comparisons among Proposed Methods

For a binary classifier, the metrics of accuracy A, recall R, precision P, and F1 score $F1$ are defined as follows:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \qquad R = \frac{TP}{TP + FN} \qquad P = \frac{TP}{TP + FP} \qquad F1 = \frac{2P \cdot R}{P + R} \tag{4}$$

where TP, FP, FN stand for True Positive, False Positive, False Negative, respectively. For anomaly detection, typically the Normal class is viewed as the negative class, and the Attack class is viewed as the positive class.

For a multiclass classifier with m classes, the metrics of accuracy A_i, recall R_i, precision P_i, and F1 score $F1_i$ are defined for each class $1 \leq i \leq m$:

$$A_i = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \qquad R_i = \frac{TP_i}{TP_i + FN_i} \qquad P_i = \frac{TP_i}{TP_i + FP_i} \qquad F1_i = \frac{2P_i \cdot R_i}{P_i + R_i} \tag{5}$$

where class i is the positive class, and all other classes in aggregate form the negative class, in a One-vs-Rest approach. For example, if class i is a KA, then the Normal class and all the other KAs are viewed as the negative class; If class i is Normal, then all the KAs are viewed as the negative class (which is counter-intuitive).

For a multiclass classifier, the macro versions of recall, precision, and F1 score are defined to measure the average performance of the classifier across each of the m classes [20] (note that there is no macro version of the accuracy metric):

$$R_{macro} = \frac{1}{m}\sum_{i}^{m} R_i \qquad P_{macro} = \frac{1}{m}\sum_{i}^{m} P_i \qquad F1_{macro} = \frac{2 * P_{macro} \cdot R_{macro}}{P_{macro} + R_{macro}} \tag{6}$$

Among all the metrics above, $F1_{macro}$ is the most comprehensive and important performance metric, since it takes into account both recall and precision of all classes.

All our four proposed methods are multiclass classifiers with $n + 2 = 5$ classes. For cascaded two-stage classification architectures, we treat the two classifiers as a single aggregate classifier. In subsequent tables showing performance results, we include the metrics for each of the 4 attack classes (3 KAs and 1 UA), computed with (5), but not those for the Normal class (which can be derived from the metrics for the 4 attack classes if needed), as well as the macro metrics, computed with (6) by averaging across all 5 classes.

The OOD threshold is an important hyper-parameter with a large impact on the IDS performance. (The decision threshold for the sigmoid function in the RFC is another hyper-parameter, which is set to be the default threshold of 0.5 after experimentation.) Fig. 9 shows how the macro F1 scores of the four methods are affected by the OOD threshold. We observe that: 1) the optimal OOD threshold setting is different for each of the three methods that are based on OOD detection; 2) different methods have different levels of sensitivity to the OOD threshold, with ACGAN-AC-RFC-Th being the least sensitive among the three. It is beneficial to be insensitive to the OOD threshold, as this reduces the burden of searching for the optimal OOD threshold at deployment time; 3) ACGAN-AC-RFC-Th has the best performance among all four methods, regardless of the OOD threshold value; 4) Since ACGAN-AC-RFC-Normal is not based on OOD detection, its F1 score is a horizontal line in Fig. 9, and the same across Tables 6 to 14.

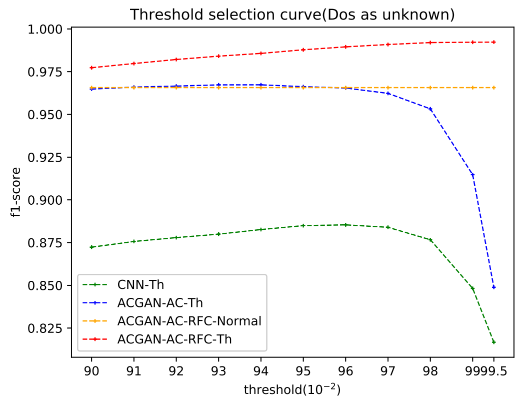


Fig. 9. Tuning the OOD threshold to maximize the macro F1 score.

Table 6 shows performance results for 4 attack classes (3 KAs: Fuzzing, GEAR, RPM and 1 UA: DoS), with optimal OOD thresholds for each of the three methods, i.e., 0.96 for CNN-Th, 0.94 for ACGAN-AC-Th; 0.995 for ACGAN-AC-RFC-Th, based on Fig. 9. (We include additional performance evaluation results in the Appendix, with the same OOD threshold for all methods.) We make the following observations:

- ACGAN-AC-RFC-Th has the best overall performance in terms of the macro metrics. It has a slightly lower F1 score (99.53%) than CNN-Th for the KA GEAR (99.66%), but the difference is small and within the range of statistical noise, and both have very good performance. This is likely because the class GEAR is inherently easy to classify.
- CNN-th has the worst overall performance in terms of the macro metrics, and especially for the UA class (emulated by DoS), with F1 score of 70.65%. This highlights the importance of having both the AC for distinguishing between Normal and KAs, and the RFC for distinguishing between real (Normal and KAs) and fake data (UA). Comparing CNN-Th with ACGAN-AC-Th, we can see that AC trained with ACGAN by itself (without the RFC) already outperforms CNN trained with Supervised Learning, thanks to the massive amounts of synthetic (fake) data generated by G. Our training dataset is artificially balanced, with an equal number of samples (5000) for each class. In practice, the training dataset is likely to be very unbalanced with limited attack data samples, and we expect the ACGAN-based approach to be even more beneficial and out-perform CNN-th, as mentioned in Section 2.5.3.

3.3.1 Performance Evaluation of RFC. Having evaluated the performance of the overall cascaded classifier, we now focus on the second-stage binary RFC, used in the two methods ACGAN-AC-RFC-Normal and ACGAN-AC-RFC-Th. Recall that the RFC's goal is to classify UA samples as fake, and all other samples (Normal and KAs) as real. Table 7 shows the Confusion Matrix definition for the RFC, where the Ground Truth (GT) positive is the UA class, and the GT negative is defined as either Normal or one of the KAs (denoted KA_i) aggregated together. Note the difference from a traditional binary classifier for anomaly detection, where the GT positive is the Attack class (anomaly), and the GT negative is the Normal class. For example, an input sample with GT class KA_i is a GT negative sample. It is an FP if it is classified as fake by RFC, hence predicted to be the UA class; it is a TN if it is classified as real by RFC, hence predicted to be either Normal in case of ACGAN-AC-RFC-Normal, or the MSP class among the $n+1$ classes (Normal and n KAs)

Table 6. Performance results with optimal OOD thresholds for each method.

Class	**Method**	**Recall(%)**	**Precision(%)**	**F1-Score(%)**
Macro	CNN-Th(th=0.96)	88.70	89.31	88.54
	ACGAN-AC-Th (th=0.94)	96.71	96.78	96.73
	ACGAN-AC-RFC-Normal	96.62	96.76	96.57
	ACGAN-AC-RFC-Th (th=0.995)	99.23	99.24	**99.23**
UA (DoS)	CNN-Th(th=0.96)	67.08	74.62	70.65
	ACGAN-AC-Th (th=0.94)	94.24	90.58	92.37
	ACGAN-AC-RFC-Normal	86.66	99.88	92.80
	ACGAN-AC-RFC-Th (th=0.995)	98.92	99.64	**99.28**
Fuzzing	CNN-Th(th=0.96)	97.12	77.72	86.34
	ACGAN-AC-Th (th=0.94)	95.82	99.32	97.54
	ACGAN-AC-RFC-Normal	97.50	92.29	94.83
	ACGAN-AC-RFC-Th (th=0.995)	98.40	99.92	**99.15**
GEAR	CNN-Th(th=0.96)	99.60	99.72	**99.66**
	ACGAN-AC-Th (th=0.94)	99.40	99.62	99.51
	ACGAN-AC-RFC-Normal	99.58	95.90	97.70
	ACGAN-AC-RFC-Th (th=0.995)	99.70	99.36	99.53
RPM	CNN-Th(th=0.96)	99.70	95.44	97.53
	ACGAN-AC-Th (th=0.94)	99.50	100.00	99.75
	ACGAN-AC-RFC-Normal	99.56	99.84	99.70
	ACGAN-AC-RFC-Th (th=0.995)	99.70	99.90	**99.80**

Table 7. Confusion matrix definition for RFC.

GT \ Predicted	Pos (fake)	Neg (real)
Pos (UA)	TP	FN
Neg (Normal or KA_i)	FP	TN

Table 8. Composition of samples passed to the RFC.

Method	# Samples	GT Pos	GT Neg	
		UA	Normal	KA_i
ACGAN-AC-RFC-Normal	9540	4393	4992	155
ACGAN-AC-RFC-Th (th=0.94)	5806	4818	845	143
ACGAN-AC-RFC-Th (th=0.96)	5995	4892	915	188
ACGAN-AC-RFC-Th (th=0.995)	9427	4994	4068	365

in case of ACGAN-AC-RFC-Th. In the latter case, it may be given the wrong label, hence should be an FP or FN from the overall cascaded classifier's perspective, yet it is still a TN from the RFC's perspective.

Table 8 shows the composition of the samples passed to the RFC, and Table 9 shows the confusion matrices and metrics for the RFC, with 3 different OOD threshold settings. (We do not show the multiclass confusion matrix for the overall classifier due to space limitations.) Table 8 can be used to confirm the statements in Table 2 in Section 2.4:

Table 9. Confusion matrices and classification metrics for RFC.

Method	TP	TN	FP	FN	Recall (%)	Precision(%)	F1 Score(%)
ACGAN-AC-RFC-Normal	4333	5142	5	60	98.63	99.88	99.26
ACGAN-AC-RFC-Th (th=0.94)	4767	977	11	51	98.94	99.77	99.35
ACGAN-AC-RFC-Th (th=0.96)	4867	1097	6	25	99.49	99.88	99.68
ACGAN-AC-RFC-Th (th=0.995)	4946	4415	18	48	99.04	99.64	99.34

- For S1 "A UA sample is more likely to be OOD than a Normal or a KA sample" and S2 "a Normal sample is more likely to be OOD than a KA sample", consider the three rows for ACGAN-AC-RFC-Th with different OOD thresholds: they all have the numeral ranking # UA samples > # Normal samples >> # KA samples, e.g., the row for ACGAN-AC-RFC-Th (th = 0.94) shows that among 5806 OOD samples, 4818 are UA samples, 845 are Normal samples, and 143 are KA samples. (Keep in mind that the test dataset is balanced with 5000 samples for each class.)
- For S3 "A UA sample is more likely to be misclassified as Normal than as one of the KAs.", consider the row for ACGAN-AC-RFC-Normal: among 5000 UA samples, 4393 are misclassified as Normal, and 607 are misclassified as one of the KAs.
- For S4 "A UA sample is more likely to be misclassified as Normal than a KA sample being misclassified as Normal", consider the row for ACGAN-AC-RFC-Normal: among the 9540 samples classified as Normal, 4393 are UA samples (misclassified); only 155 are KA samples (misclassified); the remaining 4992 are Normal samples (correctly classified).

3.3.2 Summary and Discussions. We draw the following conclusions from the experimental results:

- OOD detection alone is insufficient in identifying the UA samples, hence we cannot blindly label all OOD samples to be UA, and must rely on the second-stage RFC to classify UA samples. The OOD samples are likely to contain a mixture of Normal, KA, and UA samples (though UA samples make up the highest percentage), and a higher OOD threshold causes more input samples to be determined to be OOD across all the classes. Fortunately, the RFC has very good performance, proving the effectiveness of ACGAN training.
- Among the two-stage multiclass classification architectures, ACGAN-AC-RFC-Th is superior to ACGAN-AC-RFC-Normal, i.e., it is better to pass OOD samples (with MSP < th) than to pass samples labeled Normal to the RFC. Among the total 5000 UA samples in the test dataset, with ACGAN-AC-RFC-Normal, 4393 UA samples are misclassified as Normal and passed to RFC; The remaining 607 UA samples are misclassified as one of the KAs, hence not passed to the RFC. With ACGAN-AC-RFC-Th, much more UA samples (4818–4994 for 3 different OOD thresholds) are passed to the RFC, regardless of how the UA samples are misclassified. This explains the lower recall (86.66%) of ACGAN-AC-RFC-Normal in comparison to ACGAN-AC-RFC-Th in Table 6.
- The optimal OOD threshold for ACGAN-AC-RFC-Th is dependent on the relative performance of the AC and the RFC. If the RFC has good performance relative to the AC, then we should set a higher OOD threshold and pass more OOD samples to it, and vice versa. In other words, give more responsibility to someone you trust more. In the extreme case when the RFC has perfect performance (recall=precision=1), then the OOD threshold should be set to $th = 1.0$, causing all samples to be determined OOD and passed to the RFC. This extreme case is equivalent to reversing the order of the AC and the RFC: the RFC is applied first to filter out the UA samples labeled fake, and the samples labeled real are passed further to the second-stage AC for fine-grained classification of Normal and KAs. In practice, with $th = 1.0$, the RFC is likely to have significantly degraded precision due to the massive number of negative samples

(with GT Normal or KA_i) it receives. (We tried setting $th = 1.0$, and obtained a macro F1 score of 98.48%, slightly worse than 99.23% with $th = 0.995$, since the RFC is good, but not perfect.)

3.4 Performance Comparisons with Related Work

Among the related works listed in Table 12, a handful of papers also use the same car-hacking dataset [50] in their experiments. Table 10 shows performance comparisons of such papers that use the same dataset on the accuracy, recall, precision and F1 score, averaged across four attacks (DoS, Fuzzing, GEAR spoofing and RPM spoofing). However, the metrics of our approach may not be directly comparable with the related works, due to the differences between the dataset construction method and the dataset features used.

Table 10. Performance comparisons with related work that use the same car-hacking dataset [50]. Refer to Table 12 for more details on each method. (Note that our work (last row) performs multiclass classification and uses macro metrics defined in Eq. (6).)

Work	Method	Accuracy (%)	Recall (%)	Precision (%)	F1 (%)
Alshammari et al. (2018)	KNN [4]	97.4	96.3	94.7	93.4
Alshammari et al. (2018)	SVM [4]	96.5	95.7	95.2	93.3
Barletta et al. (2020)	XYF-K [6]	99.1	98.39	100	98.79
Olufowobi et al. (2019)	SAIDuCANT [39]	87.21	86.66	98.24	92.0
Lokman et al. (2018)	SSAE [33]	-	98.5	98.0	98.0
Song et al. (2020)	DCNN [45]	99.93	99.84	99.84	99.91
Song et al. (2021)	DCNN+Self-Sup. [43]	95.37	93.42	96.71	94.51
Ashraf et al. (2020)	LSTM-AE [5]	99.0	99.0	100	99.0
Yang et al. (2021)	MTH-IDS [57]	99.999	99.999	99.9994	99.999
Our work (macro metrics)	ACGAN	N/A	99.23	99.24	99.23

Dataset construction method: In all the related works in Table 10, the training and the test datasets each consist of four datasets, each containing a mixture of Normal messages and one type of attack, i.e., (Normal + DoS), (Normal + Fuzzing), (Normal + GEAR), (Normal + RPM). The metrics accuracy, recall, precision and F1 score are computed by averaging the corresponding metrics for binary classification of Normal vs. Attack on each dataset, as shown in Fig. 1(a). i.e., no fine-grained label is provided for each attack type. The implicit assumption is that there is at most one attack type at any given time. In contrast, our training dataset consists of a mixture of (Normal + Fuzzing + GEAR + RPM) messages, and our test dataset consists of a mixture of (Normal + Fuzzing + GEAR + RPM + DoS) messages (assuming DoS is the UA). The multiclass classifier returns one class among the 5 classes (Normal, Fuzzing, GEAR, RPM, UA), as shown in Fig. 1(c). (Refer to Section 3.1 for more details.) Hence our problem setup is more general and can handle simultaneous occurrence of multiple KAs, gives fine-grained labels for each KA, and also detects the UA. This results in a more challenging multiclass classification problem than related works, which perform binary classification for each dataset with one attack type only. Due to this difference in problem setup, the performance metrics are also necessarily different: our multiclass classifier uses the macro versions of recall, precision, and F1 score metrics defined in Eq. (6)) for a single test dataset, while related works use the the binary classification metrics defined in Eq. (4), averaged over multiple test datasets.

Dataset features used: As discussed in Section 5, it is not fair to use payload as the part of the input for the public car-hacking dataset; only the CAN ID should be used instead. Most related works in Table 10 use CAN ID + payload, except for Song et al. (2020) [45] and Song et al. (2021) [43], who are the authors that collected the original dataset; and SAIDuCANT [39], which relies on message timing instead of message content (ID or payload) for intrusion detection.

3.5 Timing Performance

In this section we perform quantitative evaluation of the timing performance, i.e., detection latency, of our best-performing method ACGAN-AC-RFC-Th. For CAN IDS with input of a sequence of messages, the detection latency consists of two factors: the amount of time for assembling a given number of consecutive messages as input to the RNN or CNN, and the execution time of the detector/classifier itself.

3.5.1 Time Window Sizes of Each CAN Image. During runtime operation, we can adopt a sliding window approach, and move the input time window (in terms of the number of consecutive messages) forward with step size 1, i.e., by one message at a time whenever a new message appears on the bus; or a non-overlapping window approach, and move the window forward each time the step size equal to the window size; or an intermediate approach of moving the window forward with step size between 1 and the window size. Fig. 10 illustrates different step sizes (assuming hypothetically a sliding window of size 4). Different step sizes lead to different tradeoffs between detection latency and runtime overhead, e.g., step size 1 results in the lowest detection latency but highest runtime overhead. Therefore, a larger time window size does not necessarily imply a longer detection latency, since we can adopt a sliding window approach with small step size. (As the choice of different sliding window approaches is orthogonal to the detection algorithm, related works typically do not specify this aspect explicitly.)

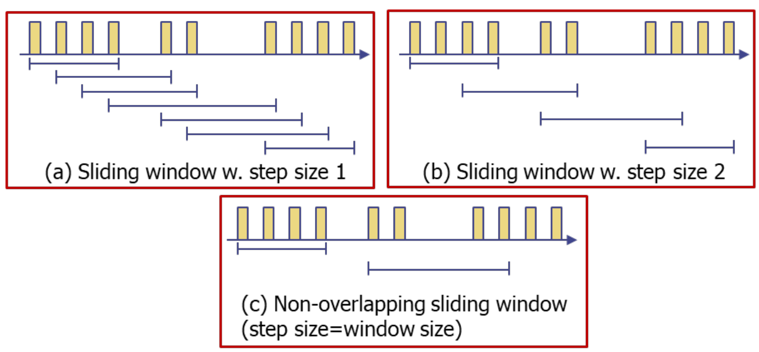


Fig. 10. Different step sizes assuming sliding window of size 4. Each box represents a CAN message.

Table 11. Time window sizes for each CAN Image of 48 messages.

Dataset	Mean (ms)	Variance
Normal	24.07	3.38
DoS	36.03	927.27
Fuzzing	36.09	1330.85
GEAR	25.72	89.41
RPM	24.76	51.04

We gathered the statistics of mean and variance for each non-overlapping window of 48 messages forming a CAN image for each type of dataset in the original public car-hacking dataset, shown in Table 11. (We also gathered the same statistics for each sliding window with step size 1, and the numbers are almost the same.) We observe that the Normal dataset has very small variance, consisting of periodic messages with period 0.5 ms. Other datasets, esp. DoS and fuzzing datasets, have large variances; and GEAR/RPM spoofing datasets have medium variances. This is consistent with our intuition of these attack types. For datasets with large time windows or large variances, it is reasonable to adopt smaller step sizes to reduce the detection latency.

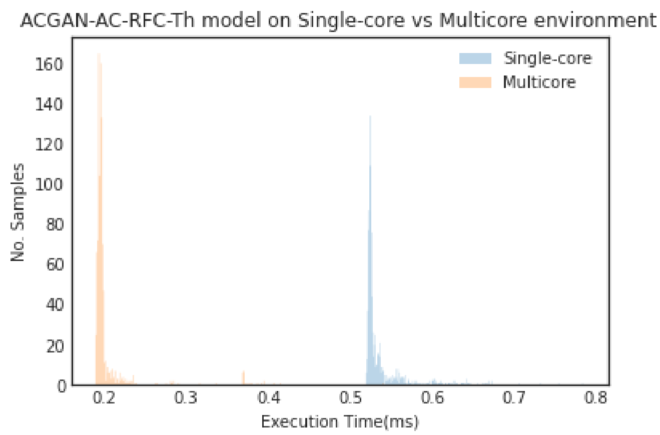


Fig. 11. Execution time distributions for single or multicore execution of the ACGAN-AC-RFC-Th model.

3.5.2 Classifier Latency and Memory Overhead. Our embedded hardware platform is Raspberry Pi 4 Model B with 8 GB memory, and quad-core ARM Cortex-A72 CPU at 1.5 GHz clock speed. The Cortex-A72 ARM CPU is representative of a medium-to-high end automotive ECU from a hardware resource and performance perspective. We adopt the TVM compiler for Deep Learning [12] to generate C code to run on the target device, which was shown to outperform hand-optimized TensorFlow Lite for well-known neural network architectures such as ResNet and MobileNet [12]. We consider both single-core execution, where the neural network inference is constrained to execute on a single core; and multi-core execution, where the neural network inference executes in parallel on all four cores. The weights are single-precision 32-bit floating point. Fig. 11 shows the execution time distributions for single or multicore execution of the ACGAN-AC-RFC-Th model. For single-core execution, the average execution time and standard deviation are 0.538 ms and 0.030 ms, respectively. For multicore execution, the average execution time and standard deviation are 0.203 ms and 0.032 ms, respectively. This is comparable to the average execution time of 0.574 ms on a Raspberry Pi 3 device with quad-core ARM Cortex-A53 CPU at 1.2 GHz clock speed for the classic ML approach in MTH-IDS [57] (it is not specified whether single-core or multicore execution is used), and well within the 10 ms latency requirement for in-vehicle IDS [2]. We also observe that multicore execution achieves significant speedup compared to single-core execution, but the speedup is much less than 4-fold on the quad-core Cortex-A72 processor.

[2]From [57]: according to the U.S. Department of Transportation, the highest priority vehicle safety services, such as collision and attack warnings, should have a latency of at most 10 to 100 ms [1]. On the other hand, for autonomous or cooperative driving, the V2X traffic safety applications require a stringent latency requirement of 10–20 ms [36]. Thus, for a vehicle-level IDS, the time needed to process each network packet is required to be less than 10 ms to meet the real-time or latency requirements.

In addition, we used the tool PyTorch-OpCounter [3] to obtain the total number of parameters of the ACGAN-AC-RFC-Th model to be 104,518, which amounts to 418.1 KB for 32-bit weights, and 104.5 KB for 8-bit weights. This is a quite small model compared to well-known models such as ResNet and MobileNet with millions of parameters, and can fit into the L2 cache of the Cortex-A72 ARM CPU, which has cache size ranging from 512 KB to 4 MB [4].

4 RELATED WORK

CAN bus IDS is an active research area due to its critical importance in automotive cyber-security [52]. The detection approaches can be broadly categorized into either rule-based, where humans design detection rules manually, or ML-based, where an ML model is trained from data. Rule-based approaches include: methods based on physical fingerprints, e.g., clock [59] or voltage measurements [14]; methods based on message timing [39, 44] or frequency [48]; methods based on message ID entropy [51] or Hamming distance [46], and many others. Due to the increasing complexity and diversity of in-vehicle network workloads, rule-based methods are generally viewed as not as accurate or flexible/adaptable as ML-based methods [56]. There are a wide variety of ML algorithms, including classic ML algorithms such as Support Vector Machines (SVM), K-Nearest Neighbor (KNN), Naive Bayes, Decision Trees, etc., and the more recent Deep Learning (DL) algorithms. Next, we present a summary and comparative analysis of ML-based approaches to CAN bus IDS. They can be broadly categorized into [11]:

- Supervised Learning for classification requires a labeled training dataset with samples of both Normal and the KAs, and learns a discriminative decision boundary that separates them, with either binary classification of Normal vs. Attack, or multiclass classification of Normal and fine-grained labels of KAs. It returns the probability $P(c \mid X)$ that the input sample X has label c, with either a Sigmoid function for binary classification, or a SoftMax function for multiclass classification. The drawback of Supervised Learning is that it cannot detect the UA class well. (We can combine Supervised learning with OOD detection to detect the UA class, e.g., CNN-Th in Fig. 8(a), but its performance is generally unsatisfactory.)
- Semi-Supervised Learning for anomaly detection requires only Normal samples during training, not labeled KA samples. Anomalies are detected as outliers that deviate significantly from Normal samples. The drawback of this approach is that cannot assign fine-grained labels to KAs.
- Hybrid approaches that combine Supervised and Unsupervised/Semi-Supervised Learning, e.g., GAN, ACGAN (as in this paper).

Deep Learning architectures based on multi-layer Neural Networks can be broadly categorized into FFNN and RNN. An FFNN by definition has no feedback or recurrent connections, and consists of multiple layers that progressively extract more abstract and higher-level representations (features), which are used by downstream classification or regression tasks. CNNs are the most popular FFNN, but there are also other types, including Multi-Layer Perceptrons (MLPs), Deep Belief Network (DBN), Capsule Networks, and Transformers. An RNN contains recurrent connections to capture the temporal dimension of the input data. Vanilla RNNs suffer from short-term memory and the vanishing gradient problem, hence Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) have been proposed to remedy their shortcomings.

CAN bus IDS is inherently a problem of detecting anomalous patterns in a time series sequence of messages, and just observing one message may not be sufficient for anomaly detection in most cases. RNNs, including its variants

[3]https://github.com/Lyken17/pytorch-OpCounter
[4]https://en.wikipedia.org/wiki/AR_Cortex-A72

Table 12. Summary of representative approaches in related works. Sup. denotes *Supervised Learning*; Semi-Sup. denotes *Semi-Supervised Learning*; Self-Sup. denotes *Self-Supervised Learning*; KNN denotes *K-Nearest Neighbors*; SVM denotes *Support Vector Machine*; XYF-K denotes *X–Y fused Kohonen network with K-means clustering (XYF–K)*; SAIDuCANT denotes *Specification-Based Automotive Intrusion Detection Using CAN Timing*; SSAE denotes *Stacked Sparse AutoEncoder*; DCNN denotes *Deep Convolutional Neural Network*; MTH-IDS denotes *Multitiered Hybrid IDS*.

Model Type	Work	Method	Input	ML Type	Classifier Type
Rule-based	Olufowobi et al. 2019 [39]	SAIDuCANT	Msg Timing	Rule-based	Bin-Class
Classic ML	Alshammari et al. 2018 [4]	KNN ∨ SVM	ID & payload	Sup.	Bin-Class
	Barletta et al. 2020 [6]	XYF-K	ID & payload	Semi-Sup.	Bin-Class
	Yang et al. 2021 [57]	Hybrid	ID & payload	hybrid	Multiclass w. UA
FFNN	Kang et al. 2016 [28]	MLP	payload	Sup.	Bin-Class
	Song et al. 2020 [45]	CNN	ID	Sup.	Bin-Class
	Song et al. 2021 [43]	CNN	ID	Self-Sup.	Bin-Class w. UA
RNN (GRU, LSTM)	Rehman et al. 2021 [40]	CNN+AGRU	ID & payload	Sup.	Bin-class
	Hossain et al. 2020 [25]	LSTM Classifier	ID & payload	Sup.	Bin ∨ Multiclass
	Taylor et al. 2016 [49]	LSTM Prediction	payload	Semi-sup.	Bin-class
Autoencoder	Lokman et al. 2018 [33]	MLP SSAE	ID & payload	Semi-sup.	Bin-Class
	Ashraf et al. 2020 [5]	LSTM AE	ID & payload	Semi-sup.	Bin-class
	Hanselmann et al. 2020 [22]	LSTM AE	ID & payload	Semi-sup.	Bin-class
	Longari et al. 2020 [34]	LSTM AE	ID & payload	Semi-sup.	Bin-class
GAN	Seo et al. 2018 [42]	GAN Classifier	ID	Hybrid	Bin-Class
	Yang et al. 2021 [58]	GAN Classifier	ID & payload	Hybrid	Multiclass
	Xie et al. 2021 [55]	GAN Classifier	ID & payload	Hybrid	Bin-Class
	Our work	GAN Classifier	ID	hybrid	Multiclass w. UA

LSTM and GRU, can be naturally applied to capture the temporal dimension of multiple messages in a time window. It is also possible to take into account the temporal dimension with a Feed-Forward Neural Network (FFNN) (including MLP and CNN) by stacking multiple messages in a time window into a 1D or 2D image as its input, as shown in Fig. 4.

Next, we discuss representative related works, summarized in Table 12. (This selection is intended to be representative, not exhaustive, and we refer the interested reader to several survey papers for more comprehensive coverage [26, 52, 56].)

4.1 Specification-based IDS

Olufowobi et al. [39] present Specification-based (i.e., rule-based) Automotive Intrusion Detection using CAN Timing (SAIDuCANT), which uses the real-time model of the CAN bus to specify intended behavior, and detect timing violations as signs of intrusions.

4.2 Classic ML-based IDS

Alshammari et al. [4] apply two classic machine learning techniques, KNN and SVM, for binary classification of Normal vs. Attack. Barletta et al. [6] present X–Y fused Kohonen network with K-means clustering (XYF–K) for binary classification. Yang et al. [57] present a multitiered hybrid IDS that incorporates a signature-based IDS and an anomaly-based IDS to detect KAs and UA, with rather complex workflow consisting of 4 stages and 12 algorithms, which may present a steep learning curve to engineers who need to apply this framework in practice to address a new problem, with different datasets and/or attack types.

4.3 FFNN-based IDS

Kang et al. [27] present an MLP-based binary classifier with two hidden layers, using 64 bits of the CAN payload as 64-dimensional input. Song et al. [45] present an IDS for detecting KAs based on an advanced CNN architecture Inception-ResNet [23] for image classification. Song et al. [43] present a self-supervised method for detecting any UA using noised pseudo normal data, which consists of two deep-learning models of the generator and the detector (the same Inception-ResNet used in [45]), which generates noised pseudo normal data and detects anomalies, respectively.

4.4 RNN-based IDS

Rehman et al. [40] present a hybrid CNN and attention-based GRU model, where the CNN extracts features as input to the downstream GRU model for classification. Hossain et al. [25] used an LSTM trained on both normal and attack messages to perform either multiclass classification with 4 labels: Normal and 3 KAs, or binary classification with 2 labels: Normal vs. Attack.

Taylor et al. [49] employ an LSTM trained on normal messages only to predict the next message's payload from a past window of messages, as shown in Fig. 12. A message is considered malicious if its payload deviates from the predicted value significantly.

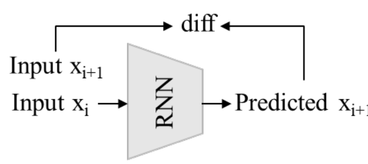


Fig. 12. RNN-based prediction.

4.5 Autoencoder-based Reconstruction

An autoencoder (AE) is trained on normal data only, and consists of an encoder that maps the input sample into the latent space representation in the bottleneck hidden layer, and a decoder (generator) that maps the representation to a reconstruction of the input sample, as shown in Fig. 13. The autoencoder is then used to compress an input sample with the encode then reconstruct it with the decoder. The reconstruction error, computed as the difference between the input sample and its reconstruction, can be used as the anomaly score. A large reconstruction error indicates a likely anomaly. The encoder and decoder in LSTM may be either FFNNs or RNNs.

Lokman et al. [33] present a MLP-based Stacked Sparse Autoencoder (SSAE) that computes the anomaly score based on the difference between an input message sequence and the reconstructed sequence. Longari et al. [34] present an LSTM-based autoencoder and an anomaly detector that works by computing the statistical characteristics of reconstruction errors over a separate validation dataset; then, it assigns a distance score that indicates how far a given reconstruction error is from the expected normal distribution. Hanselmann et al. [22] use one independent input LSTM model for each CAN, and the outputs of all input LSTM models are aggregated and fed into a fully connected subnetwork with an autoencoder structure, to take into account inter-dependencies between messages of different

IDs. Ashraf et al. [5] present an LSTM-based autoencoder, and use a statistical feature extraction technique to capture contextual features as input, instead of using the raw message bits as input.

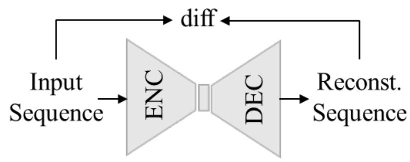


Fig. 13. Autoencoder-based Reconstruction.

4.6 GAN-based IDS

We can further divide GAN-based IDS into two types:

- GAN-based classification: use D as a binary RFC, and label the *real* input samples as Normal, and the *fake* samples as Attack.
- GAN-based reconstruction: similar to autoencoders, detect attacks as anomalies based on reconstruction errors.

4.6.1 GAN-based Classification. Seo et al. [42] present GIDS, GAN-based CAN bus IDS with cascaded binary classifier architecture, as shown in Fig. 14. The first-stage binary classier is the GAN discriminator D trained with Supervised learning (without the GAN generator G) to distinguish between Normal and one specific KA. It has good performance for the specific KA class in the training dataset, but not for other types of KAs or any UA. The second-stage binary classifier is the D, trained with GAN jointly with G, to distinguish between real (Normal) and fake (Attack) data, using only Normal samples in the training dataset. GIDS has a binary classifier in the first stage, which is only effective for the one KA_i class in its training dataset. For samples of another KA that is not KA_i, the burden falls on the second-stage RFC to classify them accurately. There is thus a dilemma of which binary classifier to use among the n binary classifiers trained with each of the n KAs, with the most logical choice being the one trained with the most frequently occurring KA_i class. We avoid this dilemma by using the multiclass AC as the first-stage classifier in ACGAN-AC-RFC-Normal and ACGAN-AC-RFC-Th.

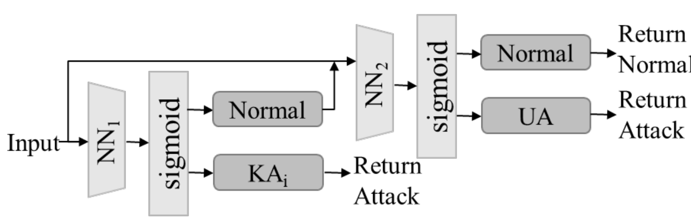


Fig. 14. GAN-based CAN bus IDS (GIDS) [42].

Yang et al. [58] present a GAN-based IDS for multiclass classification of KAs. Xie et al. [55] considered the CAN communication matrix, and grouped all messages from the same sender into a data block as input to the IDS, and achieved improved performance compared to [58].

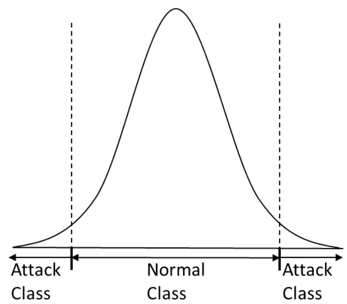
4.6.2 GAN-based Reconstruction. Donahue et al. present BiGAN (Bidirectional GAN) [17], a type of GAN where generator G not only generates data samples from the latent representation, but also includes an encoder for inverse mapping from data samples to the latent representation. This encoder enables an additional term of reconstruction loss in the GAN loss function, which can be used for anomaly detection based on the reconstruction error. Schlegl et al. present AnoGAN [41], with the loss function defined as a weighted sum of the discriminator loss and the reconstruction loss. Akcay et al. present GANomaly [3] that combines autoencoder and GAN, including the following subnetworks: an autoencoder as the generator, a real/fake classifier as the discriminator, and another encoder that compresses the generated image to its latent representation. The GAN-based reconstruction approaches have shown good performance for anomaly detection in high-resolution images, but they also have much higher runtime overhead due to the complex network architecture, hence may not be suitable for deployment in resource-constrained in-vehicle embedded systems, and we are not aware of any work on CAN bus IDS using the reconstruction-based approach.

4.7 Summary and Comparative Analysis

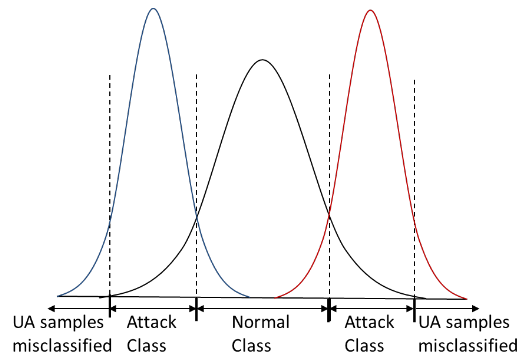
Fig. 15 compares different approaches to building a single one-stage classifier for CAN bus IDS, with the simplistic assumption of 1D Gaussian distribution for each class, showing a flatter distribution for the Normal class and sharper distributions for KA classes. Another simplistic assumption is that all UA samples are OOD, which has been shown to be not true in Section 3.3.1. The distribution of the UA class are not shown explicitly, which should have significant overlaps with that of the Normal class, based on our previous observation that "Normal samples and UA samples are more easily confused with each other, whereas each KA's samples are more easily classified and distinguished from Normal samples and UA samples". (Note that Fig. 15 does not cover the GAN-based methods in Table 12.)

- Fig. 15(a): Binary classifier trained with Semi-Supervised Learning for anomaly detection using Normal samples only. It can perform coarse-grained classification of Normal vs. Attack, i.e., any input sample that deviates from distribution of the Normal class is classified as Attack, but not fine-grained classification of KAs. Since labeled samples of KAs are not included in the training dataset, it cannot detect KAs effectively by exploiting their unique features.
- Fig. 15(b): Binary classifier trained with Supervised Learning (the two Normal distributions on the sides represent two KAs that are given the same coarse-grained label Attack.). It can perform coarse-grained classification of Normal vs. Attack, but is ineffective in fine-grained classification of KAs.
- Fig. 15(c): Multiclass classifier trained with Supervised Learning. It can perform fine-grained classification of Normal and KAs, but is ineffective in classifying UA samples.
- Fig. 15(d): Multiclass classifier trained with Supervised Learning, combined with OOD detection. It can perform fine-grained classification of Normal and KAs, and also classify UA samples. This corresponds to CNN-Th (Fig. 8(a)) and ACGAN-AC-Th (Fig. 8(b)) architectures in this paper.

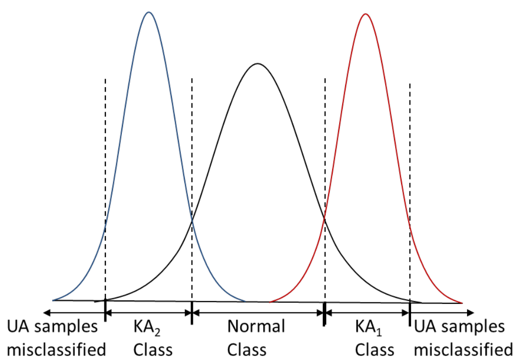
To summarize: in the presence of the UA class, a binary classifier trained with Semi-Supervised Learning (Fig. 15(a)) can be effective, but it can only perform coarse-grained classification. A binary classifier (Fig. 15(b)) or multiclass classifier (Fig. 15(c)) trained with Supervised Learning is ineffective and should not be used. Adding OOD detection (Fig. 15(d)) mitigates the problem, but is still insufficient. We assert that a two-stage cascaded architecture must be adopted for achieving both fine-grained classification of KAs and effective detection of the UA class, and the second-stage RFC for classifying the UA samples plays a crucial role in the overall classifier performance. In certain Operational Design Domains, where the attack classes encountered during operation are likely to be all KAs with no or very low chance of UA, straightforward Supervised Learning methods may be sufficient.



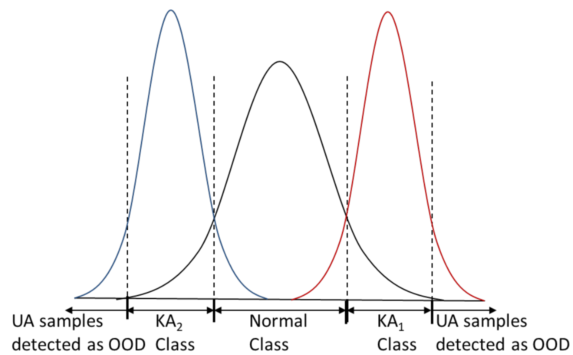
(a) Binary classifier trained with Semi-Supervised Learning for anomaly detection



(b) Binary classifier trained with Supervised Learning



(c) Multiclass classifier trained with Supervised Learning



(d) Multiclass classifier trained with Supervised Learning, with OOD Detection

Fig. 15. Intuitive illustration of different approaches to building classifiers for CAN bus IDS. The decision boundaries are visualized as vertical dotted lines.

5 DISCUSSIONS ON FEATURE SELECTION: CAN ID OR PAYLOAD

The CAN bus IDS may use CAN ID only, payload only, or both CAN ID and payload as the input feature to a classifier. The appropriate choice of input features depends on the attack scenario. Since normal CAN messages are typically periodic, this regular timing pattern of messages makes it amenable to anomaly detection based on CAN-ID only. Consider the three common types of attacks in Fig. 3: they all rely on setting fake CAN IDs for the attack messages, hence they are all detectable with CAN ID only. For example, to detect RPM spoofing attacks, we can use the CAN ID, and detect anomalies in the sequence of CAN IDs based on the periodicity of normal messages, e.g., if the normal RPM message has a period of 10ms, and we see much more frequent RPM messages within a small time window, since the spoofing attack messages are typically sent with much higher frequency than normal messages to increase the success of attack [39]. Or, we can use the payload, and detect anomalies in the time series of RPM values encoded as signals in the message payload, e.g., abrupt increase or decrease of the RPM value in a sequence of smoothly varying values.

Certain types of attacks are payload-based attacks, and the payload contains relevant information and should be used to detect them, e.g., anomalous variations in the signal values contained in the payload. But certain types of attacks are CAN ID-based attacks, e.g., for DoS, fuzzing and spoofing attacks shown in Fig. 3 and included in the car-hacking dataset [50], the CAN ID is the critical feature that distinguishes the attack messages from the normal messages. However, some related works use the payload as part of the input features for IDS, which may be problematic, since *the payloads in the public car-hacking dataset are generated randomly and form very distinguishing/discriminative features that are an artifact of the dataset generation process, and may not be present in real attacks.* Seo et al [42] describe the dataset generation procedure. The payloads of the attack messages are generated randomly, hence they are very different from the payloads of normal messages, so of course it is very effective to use the message payload as features for Normal vs. Attack classification. But this is an artifact of the dataset generation method, and real-world attacks may be very different, as a smart attacker may not use random payloads for attack messages, but instead assign similar signal values to make them look like normal message payloads, or in case of spoofing attacks, capture and replay the signal values of normal messages and use them as attack message payloads. Therefore, at least for this dataset, the message payload does not contain relevant information, and should *not* be used as input features. Methods that use the payload (e.g. [4–6, 33, 39, 57] in Table 10) have an unfair advantage over those that use CAN ID only for classification. For example, MTH-IDS by Yang et al. [57] has four features ("CAN ID," "DATA[5]," "DATA[3]," and "DATA[1]") after feature selection as the input. It is interesting to note that the authors who collected the public dataset have published 3 papers themselves using this dataset [42, 43, 45], and all of them used CAN ID only, which is the correct approach.

Even for cases where both CAN ID and payload contain relevant information, using the CAN ID only for detection has the following benefits:

- It incurs lower runtime overhead since the payload does not need to be decoded, which is important for resource-constrained in-vehicle systems;
- It is independent of the message format/payload length, hence is applicable to both the classic CAN and the more recent higher-bandwidth CAN bus stands without change, e.g., CAN-FD with payload size of 64 Bytes [16], and CAN-XL with up to 2048 Bytes [37]. The larger payload size, esp. for CAN-XL, may render many current payload-based methods ineffective, since the large payload field may overwhelm the CAN ID field in the input feature space, so it may be necessary to reduce the feature dimension by feature engineering.
- Payload-based detection is likely to be application-specific. Consider RPM spoofing attacks: the normal range and pattern of RPM values may be specific to each user/driver, and the classifier may need to be retrained/fine-tuned for different user behavior profiles, i.e., a classifier for detecting RPM spoofing attacks that is trained on a dataset of message payloads from a defensive driver may not work well on a test dataset from a race car driver, since the RPM signal may have very different patterns of variation for them. On the other hand, CAN ID-based detection is more application-neutral, and more robust to user behavior variations.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we address the problem of CAN bus intrusion detection, with the objective of detecting both KAs and any UA, and also assign fine-grained labels to detected KAs. Performance evaluation demonstrates the effectiveness of our proposed methods, especially ACGAN-AC-RFC-Th, a cascaded two-stage classification architecture, with the multi-class AC in the first stage for classification of Normal and KAs, passing OOD samples to the binary RFC in the second stage for detection of the UA class. Performance evaluation demonstrate that our method achieves both high classification accuracy and low runtime overhead. It is also conceptually simple and user-friendly, thanks to modern Deep Learning frameworks such as PyTorch and DNN compilers such as TVM.

Recently, researchers have proposed an emerging class of stealthy attacks that present significant challenges to current IDSes. Cho and Shin [13] present the bus-off attack, which exploits the error-handling mechanism of the CAN bus to shut down victim ECUs. Bloom [7] present WeepingCAN, a refinement of the bus-off attack that is more stealthy and can evade detection. Kulandaivel et al. [30] present CANnon, which leverages the peripheral clock gating feature to insert arbitrary bits at any time instance. A future research topic is the detection and mitigation of this type of stealthy attacks, which may require secure network architectures [8], physical-layer IDS, and secure transceivers.

7 ACKNOWLEDGEMENTS

We would like to thank the authors of [42] for helpful discussions of their work, and for providing the public dataset for CAN bus intrusion detection [50]. The ACGAN model used in this paper is based on Erik Linder-Norén's PyTorch implementations of GANs [32], and the overall training and testing framework is based on the source code repository of Skip-GANomaly [2].

REFERENCES

[1] Mohammad Y Abualhoul, Oyunchimeg Shagdar, and Fawzi Nashashibi. 2016. Visible Light inter-vehicle Communication for platooning of autonomous vehicles. In *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 508–513.

[2] Samet Akçay and Amir Atapour-Abarghouei andToby P. Breckon. 2019. Skip-GANomaly: Skip Connected and Adversarially Trained Encoder-Decoder Anomaly Detection. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[3] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. 2018. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*. Springer, 622–637.

[4] Abdulaziz Alshammari, Mohamed A Zohdy, Debatosh Debnath, and George Corser. 2018. Classification approach for intrusion detection in vehicle systems. *Wireless Engineering and Technology* 9, 4 (2018), 79–94.

[5] Javed Ashraf, Asim D Bakhshi, Nour Moustafa, Hasnat Khurshid, Abdullah Javed, and Amin Beheshti. 2020. Novel deep learning-enabled LSTM autoencoder architecture for discovering anomalous events from intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems* 22, 7 (2020), 4507–4518.

[6] Vita Santa Barletta, Danilo Caivano, Antonella Nannavecchia, and Michele Scalera. 2020. A Kohonen SOM architecture for intrusion detection on in-vehicle communication networks. *Applied Sciences* 10, 15 (2020), 5062.

[7] Gedare Bloom. 2021. WeepingCAN: A Stealthy CAN Bus-off Attack. In *Workshop on Automotive and Autonomous Vehicle Security (AutoSec)*, Vol. 2021. 25.

[8] Gedare Bloom, Gianluca Cena, Ivan Cibrario Bertolotti, Tingting Hu, and Adriano Valenzano. 2017. Supporting security protocols on CAN-based networks. In *2017 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 1334–1339.

[9] Saikiran Bulusu, Bhavya Kailkhura, Bo Li, Pramod K Varshney, and Dawn Song. 2020. Anomalous example detection in deep learning: A survey. *IEEE Access* 8 (2020), 132330–132347.

[10] Samarjit Chakraborty, Mohammad Abdullah Al Faruque, Wanli Chang, Dip Goswami, Marilyn Wolf, and Qi Zhu. 2016. Automotive cyber-physical systems: A tutorial introduction. *IEEE Design & Test* 33, 4 (2016), 92–108.

[11] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407* (2019).

[12] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An Automated {End-to-End} Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.

[13] Kyong-Tak Cho and Kang G Shin. 2016. Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1044–1055.

[14] Kyong-Tak Cho and Kang G Shin. 2016. Fingerprinting electronic control units for vehicle intrusion detection. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 911–927.

[15] The AUTOSAR Consortium. 2020. Requirements on Intrusion Detection System. R20-11. Foundation. AUTOSAR. Nov. 2020. https://www.autosar.org/fileadmin/user_upload/standards/foundation/20-11/AUTOSAR_RS_IntrusionDetectionSystem.pdf. Accessed: 2022-04.

[16] Ricardo De Andrade, Kleber N Hodel, Joao Francisco Justo, Armando M Laganá, Max Mauro Santos, and Zonghua Gu. 2018. Analytical and experimental performance evaluations of CAN-FD bus. *IEEE Access* 6 (2018), 21287–21295.

[17] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. 2016. Adversarial feature learning. *arXiv preprint arXiv:1605.09782* (2016).

[18] Hristos Giannopoulos, Alexander M Wyglinski, and Joseph Chapman. 2017. Securing vehicular controller area networks: An approach to active bus-level countermeasures. *IEEE Vehicular Technology Magazine* 12, 4 (2017), 60–68.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

[20] Margherita Grandini, Enrico Bagli, and Giorgio Visani. 2020. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756* (2020).

[21] Zonghua Gu, Gang Han, Haibo Zeng, and Qingling Zhao. 2016. Security-aware mapping and scheduling with hardware co-processors for FlexRay-based distributed embedded systems. *IEEE Transactions on parallel and distributed systems* 27, 10 (2016), 3044–3057.

[22] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. 2020. CANet: An unsupervised intrusion detection system for high dimensional CAN bus data. *IEEE Access* 8 (2020), 58194–58205.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[24] Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).

[25] Md Delwar Hossain, Hiroyuki Inoue, Hideya Ochiai, Doudou Fall, and Youki Kadobayashi. 2020. Long short-term memory-based intrusion detection system for in-vehicle controller area network bus. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 10–17.

[26] Hyo Jin Jo and Wonsuk Choi. 2021. A Survey of Attacks on Controller Area Networks and Corresponding Countermeasures. *IEEE Transactions on Intelligent Transportation Systems* (2021).

[27] Min-Joo Kang and Je-Won Kang. 2016. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one* 11, 6 (2016), e0155781.

[28] Min-Ju Kang and Je-Won Kang. 2016. A novel intrusion detection method using deep neural network for in-vehicle network security. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*. IEEE, 1–5.

[29] Shu Kong and Deva Ramanan. 2021. OpenGAN: Open-Set Recognition via Open Data Generation. *arXiv preprint arXiv:2104.02939* (2021).

[30] Sekar Kulandaivel, Shalabh Jain, Jorge Guajardo, and Vyas Sekar. 2021. Cannon: Reliable and stealthy remote shutdown attacks via unaltered automotive microcontrollers. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 195–210.

[31] Carson Labrado and Himanshu Thapliyal. 2019. Hardware security primitives for vehicles. *IEEE Consumer Electronics Magazine* 8, 6 (2019), 99–103.

[32] Erik Linder-Norén. 2019. PyTorch implementations of Generative adversarial networks. https://github.com/eriklindernoren/PyTorch-GAN. Accessed: 2022-04.

[33] Siti Farhana Lokman, Abu Talib Othman, Muhamad Husaini Abu Bakar, and Rizal Razuwan. 2018. Stacked sparse autoencodersbased outlier discovery for in-vehicle controller area network (CAN). *Int. J. Eng. Technol* 7, 4.33 (2018), 375–380.

[34] Stefano Longari, Daniel Humberto Nova Valcarcel, Mattia Zago, Michele Carminati, and Stefano Zanero. 2020. CANnolo: An anomaly detection system based on LSTM autoencoders for controller area network. *IEEE Transactions on Network and Service Management* (2020).

[35] Siyu Luan, Zonghua Gu, Leonid B Freidovich, Lili Jiang, and Qingling Zhao. 2021. Out-of-distribution detection for deep neural networks with isolation forest and local outlier factor. *IEEE Access* 9 (2021), 132980–132989.

[36] Abdallah Moubayed and Abdallah Shami. 2020. Softwarization, virtualization, & machine learning for intelligent & effective v2x communications. *arXiv preprint arXiv:2006.04595* (2020).

[37] Arthur Mutter. 2021. SIG CAN XL. https://www.can-cia.org/groups/technical-groups/technical-committee-tc/ig-lower-layers/sig-can-xl/. Accessed: 2022-03.

[38] Augustus Odena, Christopher Olah, and Jonathon Shlens. 2017. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*. PMLR, 2642–2651.

[39] Habeeb Olufowobi, Clinton Young, Joseph Zambreno, and Gedare Bloom. 2019. SAIDuCANT: Specification-based automotive intrusion detection using controller area network (CAN) timing. *IEEE Transactions on Vehicular Technology* 69, 2 (2019), 1484–1494.

[40] Abdul Rehman, Saif Ur Rehman, Mohibullah Khan, Mamoun Alazab, and Thippa Reddy. 2021. CANintelliIDS: detecting in-vehicle intrusion attacks on a controller area network using CNN and attention-based GRU. *IEEE Transactions on Network Science and Engineering* (2021).

[41] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. 2017. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International Conference on Information Processing in Medical Imaging*. Springer, 146–157.

[42] Eunbi Seo, Hyun Min Song, and Huy Kang Kim. 2018. Gids: Gan based intrusion detection system for in-vehicle network. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 1–6.

[43] Hyun Min Song and Huy Kang Kim. 2021. Self-supervised anomaly detection for in-vehicle network using noised pseudo normal data. *IEEE Transactions on Vehicular Technology* 70, 2 (2021), 1098–1108.

[44] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. 2016. Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network. In *2016 international conference on information networking (ICOIN)*. IEEE, 63–68.

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

[46] Dario Stabili, Mirco Marchetti, and Michele Colajanni. 2017. Detecting attacks to internal vehicle networks through Hamming distance. In *2017 AEIT International Annual Conference*. IEEE, 1–6.

[47] Christopher Szilagyi and Philip Koopman. 2009. Flexible multicast authentication for time-triggered embedded control network applications. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE, 165–174.

[48] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. 2015. Frequency-based anomaly detection for the automotive CAN bus. In *2015 World Congress on Industrial Control Systems Security (WCICSS)*. IEEE, 45–49.

[49] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. 2016. Anomaly detection in automobile control network data with long short-term memory networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 130–139.

[50] Korea University. 2018. Car-Hacking Dataset for the intrusion detection. https://ocslab.hksecurity.net/Datasets/car-hacking-dataset. Accessed: 2022-04.

[51] Wufei Wu, Yizhi Huang, Ryo Kurachi, Gang Zeng, Guoqi Xie, Renfa Li, and Keqin Li. 2018. Sliding window optimized information entropy analysis method for intrusion detection on in-vehicle networks. *IEEE Access* 6 (2018), 45233–45245.

[52] Wufei Wu, Renfa Li, Guoqi Xie, Jiyao An, Yang Bai, Jia Zhou, and Keqin Li. 2019. A survey of intrusion detection for in-vehicle networks. *IEEE Transactions on Intelligent Transportation Systems* 21, 3 (2019), 919–933.

[53] Guoqi Xie, Renfa Li, and Shiyan Hu. 2020. Security-aware obfuscated priority assignment for CAN-FD messages in real-time parallel automotive applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 4413–4425.

[54] Guoqi Xie, Laurence T Yang, Yao Liu, Haibo Luo, Xin Peng, and Renfa Li. 2021. Security Enhancement for Real-Time Independent In-Vehicle CAN-FD Messages in Vehicular Networks. *IEEE Transactions on Vehicular Technology* 70, 6 (2021).

[55] Guoqi Xie, Laurence T Yang, Yuanda Yang, Haibo Luo, Renfa Li, and Mamoun Alazab. 2021. Threat Analysis for Automotive CAN Networks: A GAN Model-Based Intrusion Detection Technique. *IEEE Transactions on Intelligent Transportation Systems* (2021).

[56] Yong Xie, Yu Zhou, Jing Xu, Jian Zhou, Xiaobai Chen, and Fu Xiao. 2021. Cybersecurity protection on in-vehicle networks for distributed automotive cyber-physical systems: State-of-the-art and future challenges. *Software: Practice and Experience* (2021).

[57] Li Yang, Abdallah Moubayed, and Abdallah Shami. 2021. MTH-IDS: A Multitiered Hybrid Intrusion Detection System for Internet of Vehicles. *IEEE Internet of Things Journal* 9, 1 (2021), 616–632.

[58] Yuanda Yang, Guoqi Xie, Jilong Wang, Jia Zhou, Ze Xia, and Renfa Li. 2021. Intrusion Detection for In-vehicle Network by Using Single GAN in Connected Vehicles. *Journal of Circuits, Systems and Computers* 30, 01 (2021), 2150007.

[59] Jia Zhou, Guoqi Xie, Siyang Yu, and Renfa Li. 2020. Clock-Based Sender Identification and Attack Detection for Automotive CAN Network. *IEEE Access* 9 (2020), 2665–2679.

A ADDITIONAL EXPERIMENTAL RESULTS

We present additional experimental results with different settings of the OOD threshold th, with $th = 0.96$ (the optimal OOD threshold for CNN-Th) in Table 13, and $th = 0.94$ (the optimal OOD threshold for ACGAN-AC-Th) in Table 14. (We use references to Fig. 8 to replace the method names to save space.) Even though they are not the optimal OOD threshold setting for ACGAN-AC-RFC-Th, it still achieves the highest macro F1 score for both threshold values. This is consistent with the results shown in Fig. 9.

Table 13. Performance results with $th = 0.96$, the optimal OOD threshold for CNN-Th.

Class	Method	Recall(%)	Precision(%)	F1-Score(%)
Macro	Fig. 8(a)	88.70	89.31	88.54
	Fig. 8(b)	96.50	96.70	96.54
	Fig. 8(c)	96.62	96.76	96.57
	Fig. 8(d)	98.95	98.98	**98.95**
UA (DoS)	Fig. 8(a)	67.08	74.62	70.65
	Fig. 8(b)	96.18	88.30	92.07
	Fig. 8(c)	86.66	99.88	92.80
	Fig. 8(d)	97.34	99.88	**98.59**
Fuzzing	Fig. 8(a)	97.12	77.72	86.34
	Fig. 8(b)	93.54	99.81	96.57
	Fig. 8(c)	97.50	92.29	94.83
	Fig. 8(d)	98.46	99.70	**99.07**
GEAR	Fig. 8(a)	99.60	99.72	**99.66**
	Fig. 8(b)	99.36	99.90	99.63
	Fig. 8(c)	99.58	95.90	97.70
	Fig. 8(d)	99.66	99.34	99.50
RPM	Fig. 8(a)	99.70	95.44	97.53
	Fig. 8(b)	99.46	100.00	99.73
	Fig. 8(c)	99.56	99.84	99.70
	Fig. 8(d)	99.66	99.94	**99.80**

Table 14. Performance results with $th = 0.94$, the optimal OOD threshold for ACGAN-AC-Th.

Class	Method	Recall(%)	Precision(%)	F1-Score(%)
Macro	Fig. 8(a)	88.63	89.25	88.26
	Fig. 8(b)	96.71	96.78	96.73
	Fig. 8(c)	96.62	96.76	96.57
	Fig. 8(d)	98.57	98.60	**98.57**
UA (DoS)	Fig. 8(a)	59.94	79.03	68.18
	Fig. 8(b)	94.24	90.58	92.37
	Fig. 8(c)	86.66	99.88	92.80
	Fig. 8(d)	95.34	99.77	**97.50**
Fuzzing	Fig. 8(a)	97.44	74.40	84.38
	Fig. 8(b)	95.82	99.32	97.54
	Fig. 8(c)	97.50	92.29	94.83
	Fig. 8(d)	98.80	97.98	**98.39**
GEAR	Fig. 8(a)	99.60	99.68	**99.64**
	Fig. 8(b)	99.40	99.62	99.51
	Fig. 8(c)	99.58	95.90	97.70
	Fig. 8(d)	99.68	99.11	99.39
RPM	Fig. 8(a)	99.70	94.07	96.81
	Fig. 8(b)	99.50	100.00	99.75
	Fig. 8(c)	99.56	99.84	99.70
	Fig. 8(d)	99.64	99.94	**99.79**