Computers & Security 140 (2024) 103777

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

`can-train-and-test`: A curated CAN dataset for automotive intrusion detection

Brooke Lampe, Weizhi Meng [*]

Technical University of Denmark, Anker Engelunds Vej 101, Kongens Lyngby, 2800, Denmark

A R T I C L E I N F O

Keywords:
Automotive security
Automotive attacks
Controller area network (CAN)
In-vehicle network (IVN)
Intrusion detection system (IDS)
Logging
Spoofing
Dataset
Benchmark
Machine learning

A B S T R A C T

When it comes to in-vehicle networks (IVNs), the controller area network (CAN) bus dominates the market; automobiles manufactured and sold worldwide depend on the CAN bus for safety-critical communications between various components of the vehicle (e.g., the engine, the transmission, the steering column). Unfortunately, the CAN bus is inherently insecure; in fact, it completely lacks controls such as authentication, authorization, and confidentiality (i.e., encryption). Therefore, researchers have travailed to develop automotive security enhancements. The automotive intrusion detection system (IDS) is especially popular in the literature—due to its relatively low cost in terms of money, resource utilization, and implementation effort. That said, developing and evaluating an automotive IDS is often challenging; if researchers do not have access to a test vehicle, then they are forced to depend on publicly available CAN data—which is not without limitations. Lack of access to adequate CAN data, then, becomes a barrier to entry into automotive security research.

We seek to lower that barrier to entry by introducing a new CAN dataset to facilitate the development and evaluation of automotive IDSs. Our datasets—dubbed `can-dataset`, `can-log`, `can-csv`, `can-ml`, and `can-train-and-test`—provide CAN data from four different vehicles produced by two different manufacturers. The attack captures for each vehicle model are equivalent, enabling researchers to assess the ability of a given IDS to generalize to different vehicle models and even different vehicle manufacturers. Our datasets contain replayable `.log` files as well as labeled and unlabeled `.csv` files, thereby meeting a variety of development and evaluation needs. In particular, the `can-train-and-test` dataset offers nine unique attacks, ranging from denial of service (DoS) to gear spoofing to standstill; as such, researchers can select a subset of the attacks for training and save the remainder for testing in order to assess a given IDS against unseen attacks. Many of our attacks, particularly the spoofing-related attacks, were conducted during live, on-the-road experiments with real vehicles. These attacks have known *physical* impacts. As a benchmark, we pit a number of machine learning IDSs against our dataset and analyze the results. We present our datasets—especially `can-train-and-test`—as a contribution to the existing catalogue of open-access datasets in hopes of filling in the gaps left by those datasets.

1. Introduction

The modern automobile has shifted from essentially mechanical to markedly electronic. Electronic control units (ECUs) now manage automotive components from the engine to the transmission to the steering column to the airbags. Modern passenger vehicles routinely contain over 80 ECUs (Novotná, 2023); when it comes to luxury vehicles, which often incorporate a number of autonomous and semi-autonomous features, the number of ECUs can be much higher (Zalman, 2017; Novotná, 2023).

ECUs communicate via in-vehicle networks (IVNs); the controller area network (CAN) is by far the most popular. The CAN bus, depicted in Fig. 1, facilitates reliable multiplex communication across twisted-pair electrical wiring. It leverages differential signaling to minimize electrical interference (i.e., noise). In the absence of an attack, it is a highly robust protocol; however, it is fundamentally insecure. Modern-day security practices—authentication, authorization, confidentiality (i.e., encryption)—have never been incorporated into the CAN bus. While the CAN bus does have integrity controls when it comes to error-handling, it offers no integrity in an attack scenario (Bozdal et al., 2020; Lampe and Meng, 2023c).

* Corresponding author.
E-mail addresses: blam@dtu.dk (B. Lampe), weme@dtu.dk (W. Meng).

https://doi.org/10.1016/j.cose.2024.103777
Received 17 June 2023; Received in revised form 17 January 2024; Accepted 18 February 2024
Available online 23 February 2024

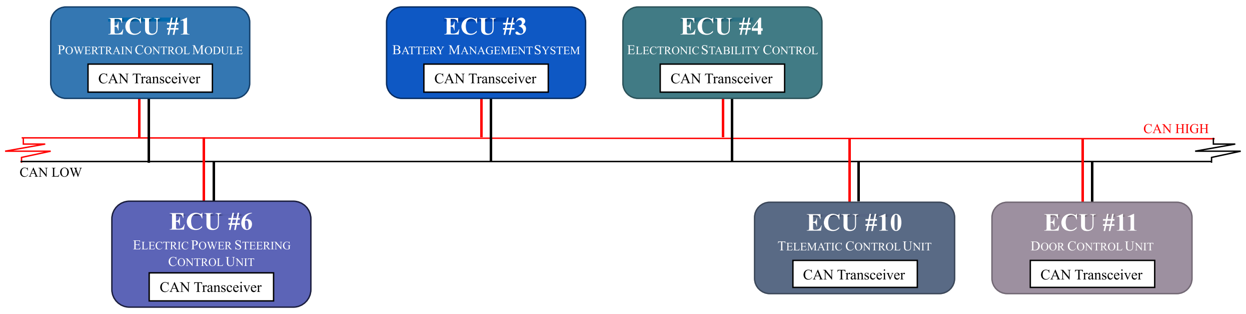

Fig. 1. Schematic diagram of the CAN bus.

Researchers have identified a number of attacks against the CAN bus, which can be organized into six families (Lampe and Meng, 2023b). These six attack families are enumerated in Table 1. Many of the attacks are simple—even trivial—to implement, and the impacts range from irritating to potentially lethal. Multiple CAN-related exploits have been published (see Table 2); fortunately, to our knowledge, no automotive attacks have been conducted in the wild.

The automotive intrusion detection system (IDS) has emerged as a favorite when it comes to CAN bus security. The automotive IDS is a relatively inexpensive solution—in terms of monetary cost, resource cost, and implementation cost. Re-engineering effort is a major roadblock to cryptographic CAN bus security solutions (e.g., authentication, encryption). Generally speaking, automotive intrusion detection systems do not engender a redesign of existing controller area networks (Lampe and Meng, 2023c). In fact, automotive IDSs do not need to be integrated into the vehicle during manufacturing; they can be deployed at any time during the vehicle's service life. Some automotive IDSs can even be deployed by ordinary consumers with no mechanical or technical expertise whatsoever (Lampe and Meng, 2022).

1.1. Motivation

In this section, we justify our motivation; specifically, we highlight the limitations of existing open-access CAN bus datasets. As mentioned earlier, intrusion detection systems are exceedingly valuable to automotive security and to automotive security research. However, automotive IDSs depend on automotive data—i.e., CAN data—for development, experimentation, and evaluation. If researchers do not have access to a test vehicle, then the lack of sufficient CAN data becomes a barrier to entry into automotive security research. Therefore, we curated a new CAN dataset, `can-train-and-test`, to address the limitations of existing open-access datasets.

We noted the following limitations of existing publicly available datasets:

1. Lack of sufficient attack-free data
2. Lack of sufficient attack variation
3. Lack of sufficient vehicle variation
4. Lack of fidelity
5. Lack of severity
6. Lack of modernity
7. Lack of labels

Lack of sufficient attack-free data. When training a machine learning IDS, especially a *deep* learning IDS, attack-free ("normal") data is crucial. Attack-free data provides a machine learning-based IDS with a much-needed baseline; once the IDS has established a baseline, it can flag deviations from the baseline as potential attacks. Many publicly available datasets provide insufficient attack-free data to train an effective machine learning classifier.

Rajapaksha et al. (2023) determined that the attack-free traffic captures relevant to each vehicle in the HCRL Survival Analysis dataset (Han et al., 2018) are not large enough to properly train a machine learning model—let alone a deep learning model.

Lack of sufficient attack variation. Generally speaking, automotive IDSs are envisaged as anomaly-based IDSs, meaning that they classify traffic as either (1) "normal" or (2) "anomalous." Typically, they build a profile of "normal" traffic during training. During testing, if they detect a deviation from the "normal" baseline, then they flag it as "anomalous." Anomaly-based IDSs are expected to detect previously unseen attacks. If an automotive dataset contains exactly one type of attack, then an anomaly-based IDS cannot be trained on one attack and tested on a second attack. If an automotive dataset contains no attacks at all, then it cannot be used to even train—let alone test—a supervised anomaly-based IDS.

The more attacks a dataset contains, the more options researchers have for training and testing with different combinations of known and unknown attacks. Therefore, when constructing our dataset, we crafted a variety of unique, realistic automotive attacks.

Several datasets (Kang and Kang, 2016; Sami et al., 2020; Sami, 2019) contain only one or two attacks; several others (CrySyS Lab, 2017; Kaiser et al., 2019; Song and Kim, 2020; Zago et al., 2020b,a; University of Turku, 2021) lack attack data altogether.

Lack of sufficient vehicle variation. The CAN protocol is ubiquitous, but the implementation is not. Different automotive manufacturers use different CAN messages to communicate different information. As such, an automotive IDS trained and tested against one type of vehicle might not generalize to vehicles produced by a different manufacturer. Such an IDS might not even generalize to different vehicle models produced by the same manufacturer. Thus, an automotive dataset should include the same types of attacks against different vehicles, allowing researchers to assess the detection capabilities of a proposed IDS against multiple vehicles.

Many automotive intrusion detection systems are built upon deep learning. When properly trained and optimized, such IDSs can be extremely powerful, accurate, and robust. However, training a deep learning model can be incredibly time consuming—and generally requires ample computing resources and training data. Researchers might want to train an IDS on one type of vehicle and then use it to monitor many different types of vehicles. If the IDS does not generalize well, then its performance metrics will drop—perhaps significantly—when faced with an unknown vehicle type. Section 7 demonstrates this phenomenon: when pitted against a known vehicle and a known attack, the multi-layer perceptron (MLP) model earns an F1-score of 0.9811. When pitted against a known vehicle and an *unknown attack,* it attains an F1-score of 0.9964. But, when the MLP model is pitted against a known attack and an *unknown vehicle,* its F1-score drops to 0.9594. The MLP model performed better when facing an unknown attack than when facing an unknown vehicle. Similarly, traditional machine learning models tended to perform better against unknown attacks than against unknown vehicles. The Local Outlier Factor (LOF) model earned an F1-score of 0.9465 in the known vehicle, known attack scenario, and an F1-score of 0.9665 in the known vehicle, unknown attack scenario. When faced with a known attack and an unknown vehicle, the F1-score fell to 0.2640. It is clear that vehicle variation can have a significant impact on the performance of an IDS. A potential solution would be to incorporate deep transfer learning into the IDS, which would drastically reduce the training burden while preparing the IDS to confront a new vehicle (Sun et al., 2022; Lampe and Meng, 2023c,b).

Two of the existing datasets (Kang and Kang, 2016; Hanselmann et al., 2020) were collected from a testbed; several more (CrySyS Lab, 2017; Kaiser et al., 2019; Song and Kim, 2020; Verma et al., 2020, 2022) do not disclose the model—or even the manufacturer—of the test vehicle. Still more datasets (Lee et al., 2017; Seo et al., 2018; Song et al., 2020; Stabili and Marchetti, 2019; Sami et al., 2020; Sami, 2019; Kang et al., 2021; University of Turku, 2021; Stabili et al., 2022; Pollicino et al., 2023) are representative of only one vehicle.

Lack of fidelity. A number of the existing publicly available datasets were collected during simulations, from testbed environments, or while the vehicle was stationary. When experimenting with automotive attacks, safety is a major concern, prompting researchers to focus on lower-fidelity options (e.g., simulations and testbeds). However, the goal of automotive security research is to develop solutions that will be deployed on real vehicles—not simulations and testbeds. Therefore, it is prudent to evaluate automotive security solutions with higher-fidelity datasets. If the evaluation dataset is highly realistic, then we can be more confident that a high-performing IDS will perform effectively in a real-world deployment. If, however, the evaluation dataset is highly *un*realistic, then we cannot extrapolate that a high-performing IDS will still perform effectively when faced with a much-different real-world deployment.

Two of the existing datasets (Kang and Kang, 2016; Hanselmann et al., 2020) were collected from a testbed, which raises fidelity concerns. Looking at the HCRL Car-Hacking dataset (Seo et al., 2018), Verma et al. (2020, 2022); Rajapaksha et al. (2023) discovered that the vehicle was never driven during the attacks. The Bus-Off dataset, the DAGA dataset, and the Ventus dataset all contain attack-free traffic collected from a real vehicle—a 2016 Volvo V40. In fact, all three datasets contain the same attack-free CAN traffic traces. Each dataset's attack traffic is constructed by selecting an attack-free trace and either injecting or removing CAN frames. Because all of the attacks are generated from the same attack-free traces—for all three datasets—the datasets lack fidelity. In particular, they lack variation. When training a machine learning IDS, lack of natural variability in the data can be particularly problematic, leading to overfitting. The researchers behind the Real ORNL Automotive Dynamometer CAN intrusion (ROAD) dataset (Verma et al., 2020, 2022) acknowledge that dynamometer data is known to be subtly different from CAN bus data collected during normal vehicle operation—which they cite as a limitation of their dataset.

Lack of severity. When pondering the possibility of a malicious, in-the-wild automotive attack, few people would expect the adversary to turn on the check engine light and stop there. Instead, we would expect a real-world attack to be a high-severity attack. Perhaps the adversary would turn the steering wheel 180 degrees while the vehicle was cruising down the interstate (Miller and Valasek, 2016b,a). Perhaps the adversary would deploy the airbag (Dürrwang et al., 2018). Perhaps the adversary would spoof ignition messages in order to start the vehicle and drive off with it (Tindell, 2023). In a CAN dataset, low-severity automotive attacks can be very useful; however, it is important to include high-severity attacks as well.

Lack of modernity. Some automotive manufacturers have programmed enhanced security features into the CAN bus and/or the ECUs. For example, when we experimented with a 2016 Chevrolet Silverado, we would be ejected from the CAN bus if we sent too many spoofed CAN frames at too high a frequency. In contrast, the 2011 Chevrolet Impala rarely excluded us from CAN communications. To our knowledge, few of the existing datasets include attacks which have been adapted to subvert the enhanced security features of some modern automobiles. With each publicized automotive exploit, security gains traction with automotive manufacturers; therefore, it is important to build datasets that contain attacks capable of eluding enhanced security features.

We were unable to determine the year of manufacture for the vehicles used in the datasets analyzed in this paper. However, we know that one dataset was published in 2016 (Kang and Kang, 2016), two in 2017 (CrySyS Lab, 2017; Lee et al., 2017), and two more in 2018 (Seo et al., 2018; Song et al., 2020; Han et al., 2018). It is probable that some of the vehicles involved in these datasets are older vehicles that lack enhanced security features. Upon reading the documentation related to these datasets (e.g., related publications, README files, etc.), we did not see any indication that the attacks were modified to counteract enhanced security features.

Lack of labels. Many IDSs rely on supervised machine learning, which, in turn, relies on labeled datasets. We use the term "labeled" to refer to a dataset in which the individual samples (e.g., individual CAN frames) are pre-labeled to indicate whether they are attack-free samples or attack samples. The label might be a "0" or a "1" to denote "attack-free" and "attack," respectively, or it might be a more specific label that identifies a particular type of attack. We simply check that the dataset is pre-labeled, meaning that practitioners do not need to label it themselves. Few of the existing datasets are adequately labeled; as such, they are ill-suited to training and testing supervised machine learning IDSs. Moreover, while unsupervised machine learning does not depend on labeled training data, an unsupervised machine learning IDS would typically be evaluated against labeled testing data, so that researchers could quantify its performance.

Excluding datasets which contain no attacks (and, thus, have no need of labels), we find that five of the existing datasets (Kang and Kang, 2016; Lee et al., 2017; Dupont et al., 2019; Verma et al., 2020, 2022; Hanselmann et al., 2020) are unlabeled. The ROAD dataset (Verma et al., 2020, 2022) is not pre-labeled, but it does contain metadata (e.g., `injection_id`, `injection_interval`) that would help a practitioner to label the dataset. However, if an attack does not target a particular arbitration ID, then the `injection_id` is `null`; as such, it is harder for practitioners to pinpoint the attack CAN frames and label the dataset themselves. Verma et al., who developed the ROAD dataset, cite the lack of message labels as a limitation of their dataset.

In summary, motivated by the limitations of existing datasets, we seek to supply a new CAN dataset that provides (1) ample attack-free data, (2) a variety of different attack types, (3) a variety of different vehicle types—e.g., manufacturer, model, year, (4) high-fidelity data, (5) high-severity attacks, (6) attacks capable of evading modern security enhancements, and (7) labeled data.

While we focus on the domain of automotive intrusion detection, we note that our dataset could be used to design and assess automotive security solutions beyond IDSs (e.g., firewall- and filtering-based techniques).

1.2. Contributions

Our contributions can be summarized as follows:

1. We present a new open-access CAN dataset for use in automotive intrusion detection. Our dataset includes
   - A high volume of attack-free data
   - Nine unique attacks
   - Data from four different vehicles
   - Attack-free data and attack data that was collected during live, on-the-road experiments
   - High-severity attacks
   - Attacks adapted to evade modern security enhancements
   - Labeled attack-free and attack samples
2. We compile the `can-train-and-test` dataset to help researchers develop and evaluate machine learning IDSs. The dataset is subdivided into four train/test sub-datasets: `set_01`, `set_02`,

SOF	ID	Control	DLC	Data	CRC	ACK	EOF
Start of Frame	Arbitration Identifier	Control	Data Length Code	Data	Cyclic Redundancy Checksum	Acknowledge	End of Frame
1-bit	11-bit	3-bit	4-bit	0-8 bytes	16-bit	2-bit	7-bit

Fig. 2. CAN frame specification.

set_03, set_04. Within each train/test sub-dataset, we provide one training subset and four testing subsets.

3. We curate the `can-train-and-test` dataset to optimize its assessment capabilities. The dataset provides testing subsets, which challenge an automotive IDS's ability to generalize to different attacks and vehicles:
   - **train_01:** Train the model
   - **test_01_known_vehicle_known_attack:** Test the model against a known vehicle (seen in training) and known attacks (seen in training)
   - **test_02_unknown_vehicle_known_attack:** Test the model against an unknown vehicle (not seen in training) and known attacks (seen in training)
   - **test_03_known_vehicle_unknown_attack:** Test the model against a known vehicle (seen in training) and unknown attacks (not seen in training)
   - **test_04_unknown_vehicle_unknown_attack:** Test the model against an unknown vehicle (not seen in training) and unknown attacks (not seen in training)
4. Lastly, we evaluate several machine learning IDSs on our `can-train-and-test` dataset. In our benchmark, we include both supervised and unsupervised paradigms, as well as traditional machine learning and deep learning models. Our benchmark encompasses seven model families, which contain a total of eighteen models.

1.3. Paper organization

The remainder of this paper is organized as follows: Section 2 encapsulates essential background on automotive security and controller area networks. Section 3 explores a number of related works, in particular, existing publicly available CAN datasets. Section 4 discusses our methodology (e.g., data collection, data pre-processing, etc.), while Section 5 specifies the qualitative and quantitative particulars of our datasets. Section 6 provides practitioner guidance (i.e., which dataset to use) as well as usage notes and examples. Our benchmark is detailed in depth in Section 7. Section 8 investigates the limitations of our work, while Section 9 highlights open challenges and opportunities for future work. Section 10 concludes our work.

2. Background

2.1. The protocol

The controller area network (CAN) bus is the de facto standard when it comes to in-vehicle networks (IVNs). The CAN protocol was developed by Robert Bosch GmbH (i.e., BOSCH), a German automotive control systems manufacturer, in 1983. In 1993, the protocol was adopted as an ISO standard—no. 11898 (Foster and Koscher, 2015).

The CAN protocol's design objectives were (1) low latency, (2) high throughput, and (3) reliability. Controller area networks became popular with automotive manufacturers because they drastically reduced the amount of wiring needed. The CAN bus is a dual-wire serial bus. Two wires—a twisted pair—run from the powertrain control module (PCM) to the battery management system (BMS) to the telematic control unit (TCU) and more. Every electronic control unit (ECU) in the vehicle is physically connected to the CAN bus via these two wires. Wiring is dramatically reduced; instead of directly connecting every ECU to every other ECU (and to the CAN bus) using separate wires, every ECU is connected to the two wires that comprise the CAN bus. The ECUs share the CAN bus and take turns communicating. An arbitration scheme allows the ECUs to take turns without conflict or data loss (Foster and Koscher, 2015; Bozdal et al., 2020).

Fig. 1 illustrates the CAN bus: two wires—`CAN HIGH` and `CAN LOW` interconnect various ECUs. As the names suggest, `CAN HIGH` is the high voltage wire; `CAN LOW` is the low voltage wire. The difference in voltage—not the voltages themselves—determine the signal that is communicated. If `CAN HIGH` is less than or equal to `CAN LOW`, then the recessive logic—1—is transmitted. The recessive logic is the default logic. If, instead, `CAN HIGH` is greater than `CAN LOW`, then the dominant logic—0—is transmitted. To communicate, an ECU will drive the CAN bus to the dominant state. When communication ceases, 120 Ω terminating resistors pull the CAN bus back to the recessive state. Generally, both `CAN HIGH` and `CAN LOW` are approximately 2.5 V when the bus is in the recessive state; in the dominant state, `CAN HIGH` is driven to 5 V, while `CAN LOW` is pulled to ground (Foster and Koscher, 2015; Lampe and Meng, 2023c; Lakhal et al., 2021).

As mentioned above, the difference in voltage between `CAN HIGH` and `CAN LOW` determines the signal; this property is a form of *differential signaling*. Differential signaling insulates the CAN bus from electrical interference (i.e., noise). Generally, noise will impact both `CAN HIGH` and `CAN LOW`; as such, the voltage difference will not change.

ECUs communicate via packets known as *CAN frames*. The structure of a valid CAN data frame is tightly specified, as shown in Fig. 2. The start of frame—SOF–advertises the start of transmission. The arbitration identifier—arbitration ID—determines the priority of the frame. Typically, the arbitration ID can be used to identify the transmitting ECU. The control field is comprised of several fixed control bits. The data length code—DLC—provides the size of the data field (in bytes). The data field contains the data to be communicated. One or more pieces of information will be encoded in the data field, which can range from zero to eight bytes. The cyclic redundancy checksum—CRC—is a checksum intended to confirm the data's integrity. As the name suggests, the acknowledge—ACK—field is intended for message acknowledgment; any receiving node can use the ACK field to certify that the message was received successfully. Lastly, the end of frame—EOF—demarcates the end of the transmission (Lampe and Meng, 2023c).

The CAN protocol itself is standardized across all the automotive manufacturers who have implemented it; however, the implementations are proprietary. Different manufacturers assign different arbitration IDs to different ECUs. Different manufacturers also encode different data into different CAN frames. As such, research work conducted on a particular vehicle often does not generalize to all vehicles fabricated by all manufacturers.

2.2. The threat model

A number of vulnerabilities afflict the CAN bus, many of which are correlated with four major shortcomings: (1) no authentication, (2) no authorization, (3) no encryption, and (4) no integrity. These vulnerabilities are outlined in Fig. 3.

No authentication. The CAN bus, as currently implemented, completely lacks authentication. In attack-free conditions, a given arbitration ID will generally map to a particular ECU (or node). However, a malicious node can use a legitimate ECU's arbitration ID to masquerade as the legitimate ECU. If all ECUs were required to authenticate themselves prior to transmitting, then a masquerade attack would not be as

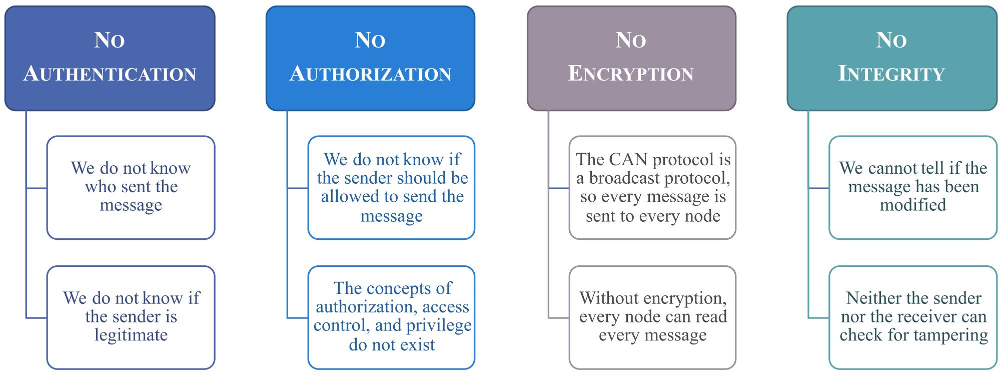
No Authentication
We do not know who sent the message
We do not know if the sender is legitimate
No Authorization
We do not know if the sender should be allowed to send the message
The concepts of authorization, access control, and privilege do not exist
No Encryption
The CAN protocol is a broadcast protocol, so every message is sent to every node
Without encryption, every node can read every message
No Integrity
We cannot tell if the message has been modified
Neither the sender nor the receiver can check for tampering

Fig. 3. Vulnerabilities of the CAN bus.

simple as using a real ECU's arbitration ID. As it stands, when a CAN frame is received, we cannot confirm that it came from the ECU we expect. We do not know if the CAN frame was sent by a legitimate ECU or a malicious node. Even if we knew that a legitimate ECU sent the CAN frame, we would not be able to determine *which* ECU was the sender.

No authorization. Authorization, as with authentication, is wholly absent from the CAN bus. In an attack-free situation, a given ECU will send CAN frames related to the information it possesses. For example, the door control unit (DCU), as the name suggests, is concerned with the functionalities provided by the doors of modern automobiles (e.g., locking and unlocking doors, rolling windows up and down, child safety locks, etc.). In attack-free conditions, we would expect the DCU to send CAN frames regarding the status of the doors—open or closed, locked or unlocked, etc. In the Chevrolet Silverado, for example, the DCU is associated with the arbitration identifier `12A`. The DCU might convey "the right front door is open" or "the left rear door is unlocked" by encoding that information in the data field of the CAN frame. The CAN frame would be sent with the arbitration ID `12A`. The DCU should *never* send CAN frames asserting the vehicle's speed, the temperature of the engine coolant, or the current gear (drive, neutral, reverse, park). If the DCU sends a CAN frame with the arbitration ID `3E9` (which communicates the vehicle's speed), then something has gone wrong. If the DCU is transmitting CAN frames that begin with the arbitration ID `3E9` and proclaim "the vehicle is traveling 15 mph," then something has gone wrong. Perhaps an attacker has compromised the DCU and is transmitting spoofed CAN frames. Without authorization, however, there is nothing to stop an ECU from transmitting information beyond its scope and function. If the concept of authorization were applied to the CAN bus, then if attackers compromised the DCU, they could only communicate DCU-related information (door and window statuses, etc.). They would not be able to spoof the vehicle's current speed, gear, or engine RPMs. Miller and Valasek demonstrated this type of attack—quite spectacularly—when they compromised a Jeep Cherokee's infotainment ECU and used it to conduct cyber-physical attacks (Miller and Valasek, 2015a,b).

Simply put, the DCU is a door control unit, and it should *only* communicate door- and window-related information. Because the CAN bus lacks authorization, attackers could compromise the DCU and spoof *any* message related to *any* vehicle function (e.g., engine RPMs, current gear). If the CAN bus enforced authorization, then the attackers would only be able to spoof DCU-related information, drastically reducing the scope of a potential attack.

No encryption. CAN traffic is transparent, unencrypted. While the information contained in the data field of a CAN frame is encoded (e.g., wheel speeds might be converted to hexadecimal and compressed), it is often trivial to decode. Data is encoded in order to maximize throughput, not ensure confidentiality. Moreover, the CAN protocol is a broadcast protocol, meaning that every message is sent to every node. If attackers attach themselves to the CAN bus or compromise even one ECU, they can read every message from every node. With a bit of reconnaissance, attackers can decode the information contained in captured CAN frames. From there, they can craft spoofed CAN frames or simply replay the captured CAN frames. If CAN traffic were encrypted, reconnaissance would be much, much harder, and the compromise of one ECU would not be nearly as catastrophic.

No integrity. As mentioned earlier, the CAN bus does have a cyclic redundancy checksum (CRC) intended to ensure data integrity. However, the CRC is effective only under attack-free conditions. If attackers wished to spoof or modify a CAN frame, they would merely need to recompute the checksum for the modified message. This computation is trivial and is part of the standard—and publicly available—CAN protocol. Therefore, under attack conditions, the CAN bus utterly lacks integrity protections. If integrity controls were incorporated into the CAN bus, we would know if the CAN frame had been subject to tampering, and the origin of the CAN frame would be verifiable.

When developing a threat model, the CIA Triad is a popular criterion. The "C," "I," and "A," refer to confidentiality, integrity, and availability, respectively. *Confidentiality* involves protecting sensitive information from unauthorized access. *Integrity* is concerned with tamper-resistance and verification of origin—we should know if the data has been altered, and, if the data has not been altered, then we should know who authored the data. *Availability* ensures that authorized entities can access the data (or service) (Daily and Van, 2021). The CAN bus falls short in terms of confidentiality, integrity, and availability. The lack of encryption results in a lack of confidentiality. The lack of integrity controls—as well as the lack of authentication—leads to a lack of integrity. Because the CAN bus lacks authentication, a malicious node can overwhelm the network with spoofed CAN frames, preventing legitimate nodes from communicating. As such, the CAN bus also lacks availability (Lampe and Meng, 2023a).

As we have seen, the CAN bus has a wide array of vulnerabilities; naturally, a wide array of CAN bus attacks have been identified and described. CAN-related attacks can be subdivided into six major attack families:

1. Denial of Service (DoS)
2. Fuzzing
3. Masquerade
4. Replay
5. Spoofing
6. Suppress

Table 1 summarizes these known CAN bus attack families.

Fig. 4 depicts the CAN bus under attack-free conditions. This diagram provides a "normal" baseline for CAN communications. We can refer back to this diagram in order to better understand the schematic diagrams of the CAN bus under various attack conditions (Figs. 5, 6, 7, 8, 9, and 10).

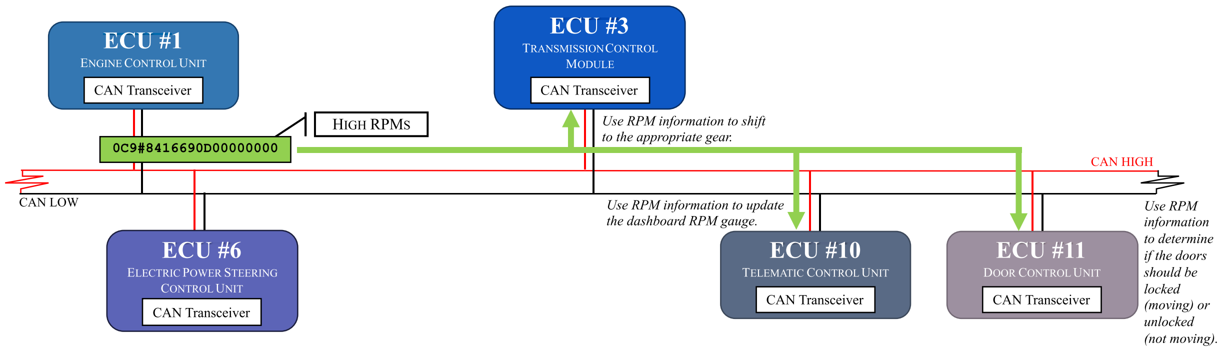
ECU #1
Engine Control Unit
CAN Transceiver
ECU #3
Transmission Control Module
CAN Transceiver
High RPMs
0C9#8416690D00000000
Use RPM information to shift to the appropriate gear.
CAN HIGH
CAN LOW
Use RPM information to update the dashboard RPM gauge.
Use RPM information to determine if the doors should be locked (moving) or unlocked (not moving).
ECU #6
Electric Power Steering Control Unit
CAN Transceiver
ECU #10
Telematic Control Unit
CAN Transceiver
ECU #11
Door Control Unit
CAN Transceiver

Fig. 4. Schematic diagram of attack-free CAN communications.

Table 1
Known CAN bus attack families.

Attack	Description	Impacts	Figure
Denial of Service (DoS)	Disable the network—i.e., inhibit legitimate ECUs from communicating over the network—by, e.g., overwhelming it with CAN frames, introducing noise, or manipulating the differential voltage. The lack of network availability constitutes a *denial of service* to legitimate ECUs.	Legitimate ECUs will revert to fail-safe parameters.	5
Fuzzing	Construct and transmit random or pseudo-random (*fuzzed*) CAN frames. An attacker might randomize both the arbitration ID and the data field, or the attacker might use known, valid arbitration IDs paired with random data fields.	Legitimate communication might be disrupted—perhaps by bus errors. Legitimate ECUs generally do not respond to invalid arbitration IDs; however, if valid arbitration IDs are paired with random data fields, then the recipient ECUs might behave erratically. Fuzzing attacks might be used as reconnaissance to prepare for subsequent masquerade or spoofing attacks.	6
Masquerade	*Masquerade* as an authentic ECU in order to, e.g., issue commands to legitimate ECUs. Often, masquerade attacks involve stifling the real ECU in order to eliminate confliction.	If deceived, legitimate ECUs could initiate dangerous—even destructive—actions.	7
Replay	Capture and *replay* authentic CAN frames, possibly as the basis for masquerade and spoofing attacks.	If deceived, legitimate ECUs could initiate dangerous—even destructive—actions.	8
Spoofing	Transmit false—i.e., *spoofed*—CAN frames to induce a reaction from legitimate ECUs.	If deceived, legitimate ECUs could initiate dangerous—even destructive—actions.	9
Suppress	Prevent a legitimate ECU from communicating over the network—i.e., *suppress* CAN frames originating from a particular ECU.	Suppressing safety-critical communications (e.g., brake-related communications) could endanger the vehicle, its occupants, and anyone nearby. Suppress attacks are often used to facilitate masquerade attacks.	10

There is considerable overlap between masquerade, replay, and spoofing attacks. In a replay attack, previously captured CAN frames are simply replayed. However, if an attacker *replays* CAN frames in order to communicate a vehicle status that is no longer true—i.e., *spoofing*—and *masquerade* as a legitimate ECU, then the attack constitutes replay, masquerade, and spoofing. Replay is a relatively simple attack; as such, it is used as a building block for the more complex attacks, e.g., masquerade and spoofing.

2.2.1. *Denial of service*

Fig. 5 illustrates the CAN bus during a denial of service (DoS) attack. As shown in the figure, the adversary transmits an overwhelming volume of high-priority CAN frames (arbitration ID `000`). CAN frames from the legitimate ECU are forced to wait on the high-priority traffic. Since the adversary transmits high-priority CAN frames so frequently, the legitimate ECU never has a chance to communicate. Thus, the legitimate ECU experiences a *denial of service*.

Our datasets (`can-dataset`, `can-ml`, and `can-train-and-test`) include DoS attacks. There are four DoS attack traces per vehicle. Generally, the 1st and 2nd DoS attack trace contain more obvious DoS attacks—e.g., an arbitration identifier `000`, a data field `0000000000000000`, and an extremely high message injection rate. We designed the 3rd and 4th traces to be more subtle—e.g., a high (but not extremely high) message injection rate, a non-zero data field, and/or a high-priority *legitimate* arbitration identifier. We discuss our attacks in detail in Section 4.3.

2.2.2. *Fuzzing*

Fig. 6 showcases the CAN bus under fuzzing conditions. The adversary is transmitting randomized CAN frames. In this example, both the arbitration ID and the data field are randomized; as such, most of the fuzzed CAN frames have meaningless arbitration IDs that will be ignored by legitimate ECUs. Legitimate ECUs listen for relevant CAN frames—relevant arbitration IDs—while ignoring CAN frames that are irrelevant.

For example, the door control unit (DCU) will listen for CAN frames bearing the transmission control module (TCM)'s arbitration ID—because the DCU wants to keep the doors locked while driving but unlock them once the vehicle parked. However, the DCU will ignore CAN frames related to fuel injection in the engine, as they are irrelevant to its function.

Therefore, while a randomized fuzzing attack can be disruptive, we would expect it to be less disruptive than a more sophisticated fuzzing attack that uses valid arbitration IDs combined with randomized data fields. This type of sophisticated fuzzing attack might randomly generate (i.e., *fuzz*) data fields that—paired with valid arbitration IDs—cause dangerous and even destructive behavior.

Similar to DoS, we include four fuzzing attack traces per vehicle. The fuzzing attacks vary in terms of message injection frequency and attack duration. Generally, the 1st and 2nd attack traffic traces are more overt (higher injection frequency, longer attack), while the 3rd and 4th are more subtle (lower injection frequency, shorter).

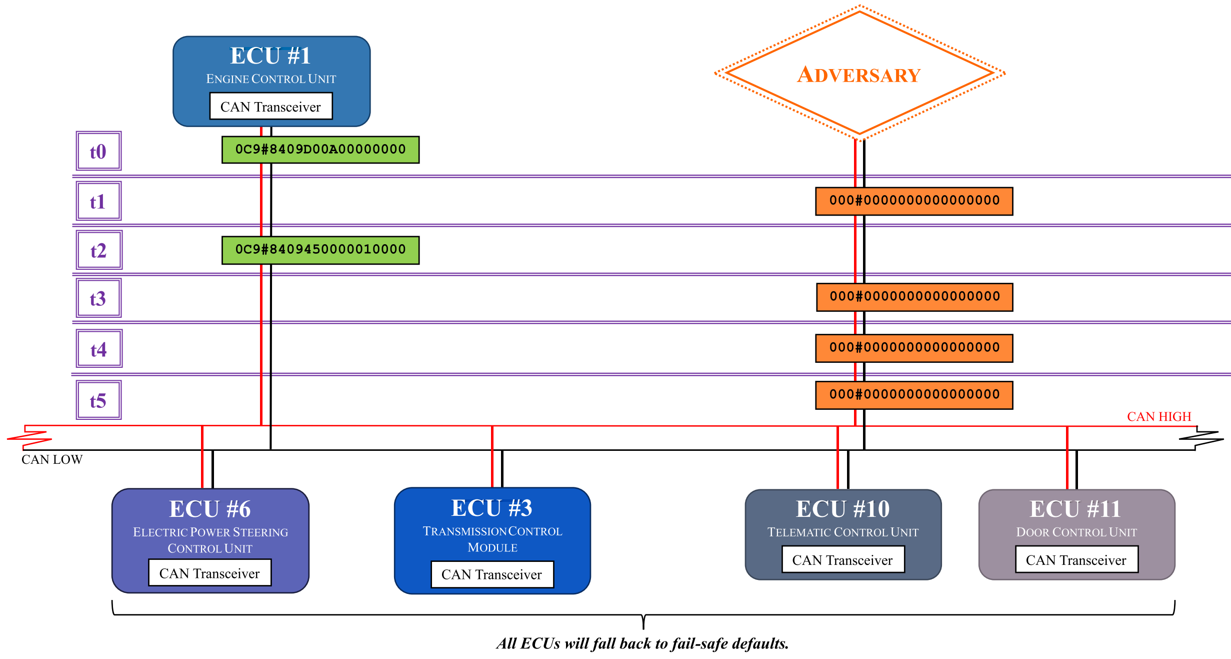

Fig. 5. Schematic diagram of a denial of service (DoS) attack.

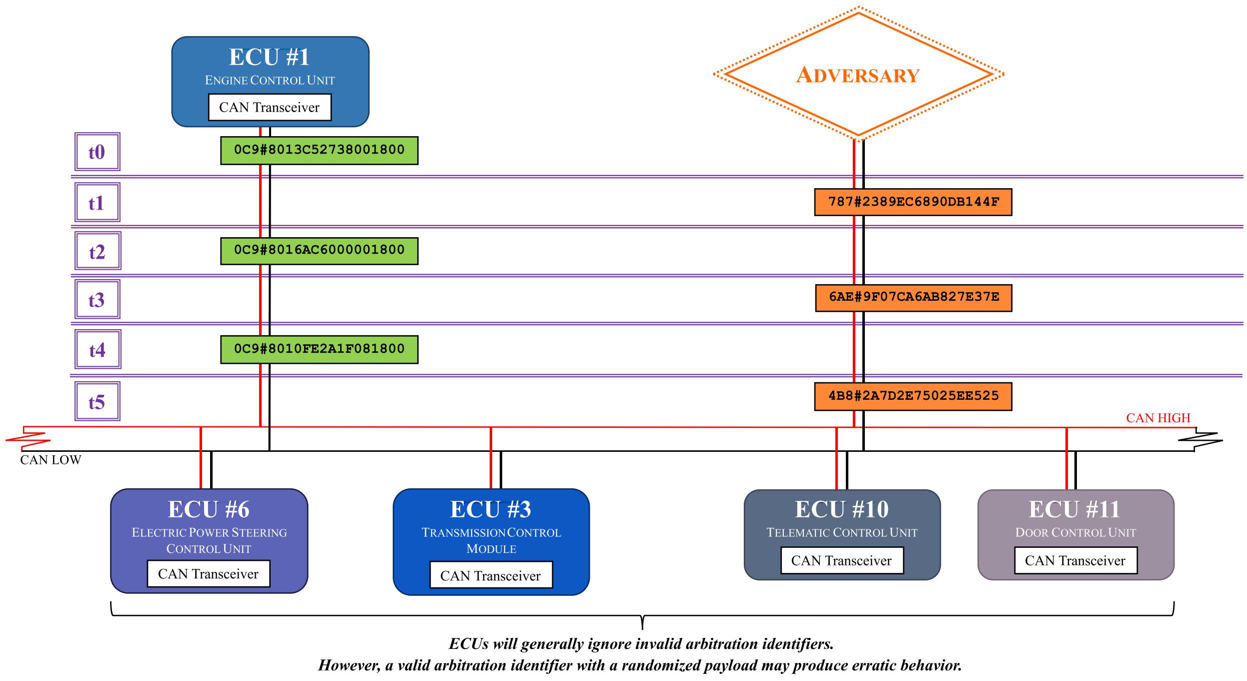

Fig. 6. Schematic diagram of a fuzzing attack.

2.2.3. Masquerade

Fig. 7 demonstrates a particularly sophisticated type of masquerade attack. A sophisticated masquerade attack combines the capabilities of replay, spoofing, and suppress attacks. A malicious node targets a particular ECU (e.g., the engine control unit). To masquerade as the legitimate ECU, the malicious node can either (1) capture and replay the legitimate ECU's CAN frames or (2) spoof CAN frames using the legitimate ECU's arbitration ID.

Of course, the replayed or spoofed CAN frames will clash with the legitimate ECU's CAN frames—this phenomenon is known as "confliction." To alleviate confliction, the malicious node must prevent the legitimate ECU from communicating, perhaps by intercepting its CAN frames. Miller and Valasek eliminated confliction by putting the legitimate ECU into Bootrom mode (by starting the reprogramming process) (Miller and Valasek, 2016b,a).

Once the legitimate ECU is silenced and the malicious node is transmitting CAN frames using the legitimate ECU's arbitration ID, the malicious node is essentially *masquerading* as the legitimate ECU. In our example, the engine control unit has been silenced; as such, the transmission control unit, the telematic control unit, and the door control

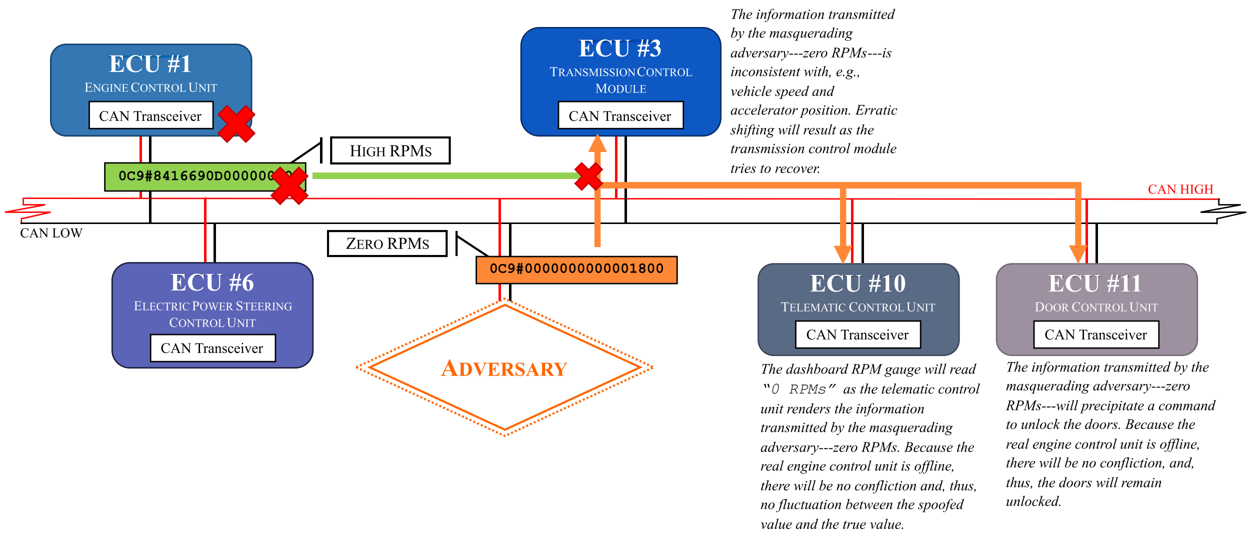
ECU #1
Engine Control Unit
CAN Transceiver
0C9#8416690D0000000
High RPMs
ECU #3
Transmission Control Module
CAN Transceiver
The information transmitted by the masquerading adversary---zero RPMs---is inconsistent with, e.g., vehicle speed and accelerator position. Erratic shifting will result as the transmission control module tries to recover.
CAN HIGH
CAN LOW
Zero RPMs
0C9#0000000000001800
ECU #6
Electric Power Steering Control Unit
CAN Transceiver
Adversary
ECU #10
Telematic Control Unit
CAN Transceiver
The dashboard RPM gauge will read "0 RPMs" as the telematic control unit renders the information transmitted by the masquerading adversary---zero RPMs. Because the real engine control unit is offline, there will be no confliction and, thus, no fluctuation between the spoofed value and the true value.
ECU #11
Door Control Unit
CAN Transceiver
The information transmitted by the masquerading adversary---zero RPMs---will precipitate a command to unlock the doors. Because the real engine control unit is offline, there will be no confliction, and, thus, the doors will remain unlocked.

Fig. 7. Schematic diagram of a masquerade attack.

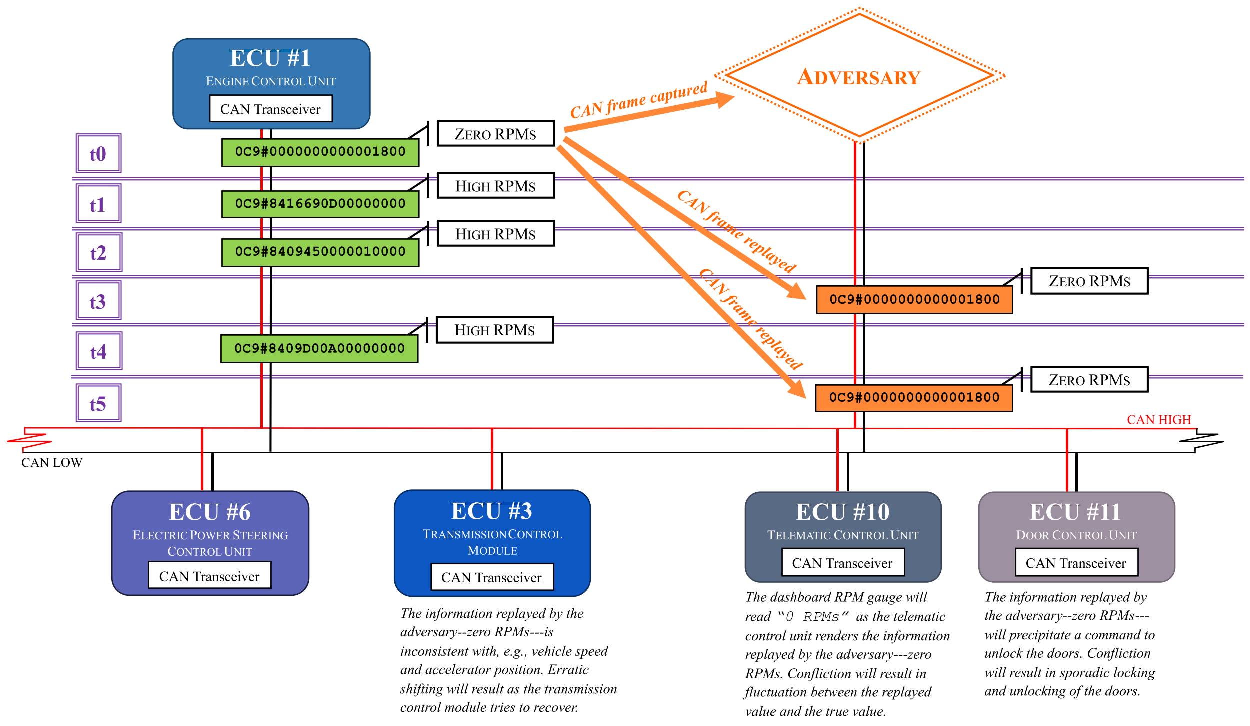
ECU #1
Engine Control Unit
CAN Transceiver
Adversary
CAN frame captured
t0
0C9#0000000000001800
Zero RPMs
t1
0C9#8416690D00000000
High RPMs
t2
0C9#8409450000010000
High RPMs
CAN frame replayed
t3
0C9#0000000000001800
Zero RPMs
t4
0C9#8409D00A00000000
High RPMs
CAN frame replayed
t5
0C9#0000000000001800
Zero RPMs
CAN HIGH
CAN LOW
ECU #6
Electric Power Steering Control Unit
CAN Transceiver
ECU #3
Transmission Control Module
CAN Transceiver
The information replayed by the adversary--zero RPMs---is inconsistent with, e.g., vehicle speed and accelerator position. Erratic shifting will result as the transmission control module tries to recover.
ECU #10
Telematic Control Unit
CAN Transceiver
The dashboard RPM gauge will read "0 RPMs" as the telematic control unit renders the information replayed by the adversary---zero RPMs. Confliction will result in fluctuation between the replayed value and the true value.
ECU #11
Door Control Unit
CAN Transceiver
The information replayed by the adversary--zero RPMs--- will precipitate a command to unlock the doors. Confliction will result in sporadic locking and unlocking of the doors.

Fig. 8. Schematic diagram of a replay attack.

unit all believe that they are receiving CAN frames from the engine control unit, but, in fact, they are receiving CAN frames from the adversary.

Our datasets include *unsophisticated* masquerade attacks, but they do not include *sophisticated masquerade attacks*. Masquerading as the speed control unit (SCU), we deceived a real vehicle, traveling at speed, into believing our spoofed CAN frames. As a result, the speedometer showed our spoofed speed rather than the true speed—in spite of the fact that the legitimate SCU was still transmitting messages. Basically, rather than conduct a sophisticated masquerade attack (by silencing the legitimate SCU), we simply overwhelmed the legitimate speed-related messages with our spoofed messages. We spoofed several different speeds, and, for each vehicle, we customized our speed spoofing messages to match the expected format. Generally, the 1st and 2nd attack traffic traces involve higher injection rates, longer attack durations, and/or multiple attacks per trace. Meanwhile, the 3rd and 4th attack traffic traces involve lower injection rates, shorter attack durations, and fewer attacks per trace—sometimes just one. This is true of all masquerade/spoofing attacks—e.g., speed spoofing, RPM spoofing, gear spoofing, etc.

2.2.4. Replay

Fig. 8 depicts a replay attack against the CAN bus. This type of attack is relatively simple; the adversary simply captures and retransmits (i.e., *replays*) CAN frames originally transmitted by legitimate ECUs. If the adversary collects a sufficient number of legitimate CAN frames, this type of attack can become incredibly powerful. In our example, the engine is generating "high RPMs" (presumably because the vehicle is in motion), but the adversary has tricked the telematic control unit and the door control unit into believing that the vehicle is stopped by trans-

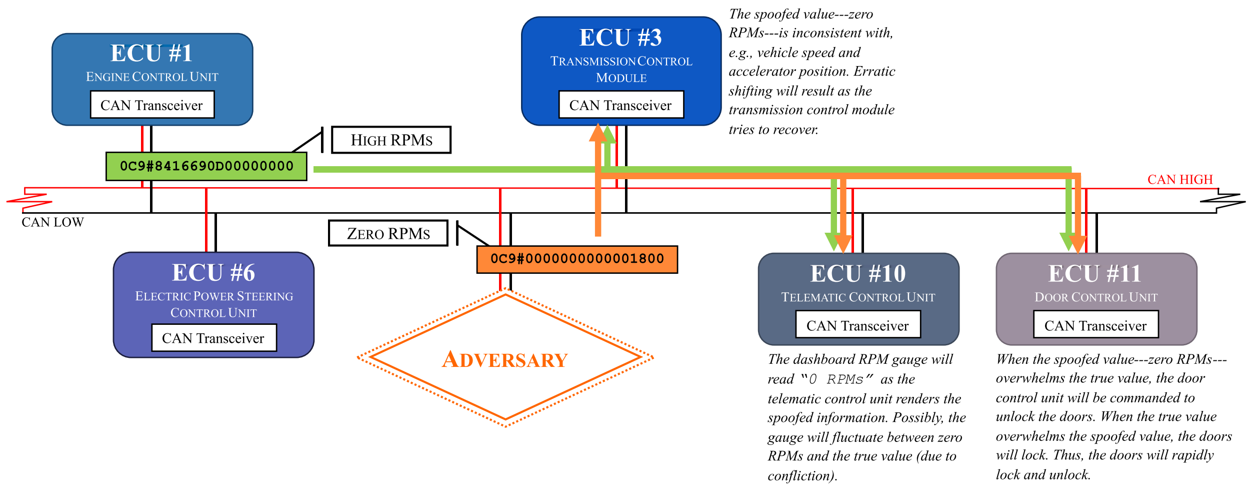
ECU #1
Engine Control Unit
CAN Transceiver
ECU #3
Transmission Control Module
CAN Transceiver
The spoofed value---zero RPMs---is inconsistent with, e.g., vehicle speed and accelerator position. Erratic shifting will result as the transmission control module tries to recover.
0C9#8416690D00000000
High RPMs
CAN HIGH
CAN LOW
Zero RPMs
0C9#0000000000001800
ECU #6
Electric Power Steering Control Unit
CAN Transceiver
Adversary
ECU #10
Telematic Control Unit
CAN Transceiver
ECU #11
Door Control Unit
CAN Transceiver
The dashboard RPM gauge will read "0 RPMs" as the telematic control unit renders the spoofed information. Possibly, the gauge will fluctuate between zero RPMs and the true value (due to confliction).
When the spoofed value---zero RPMs---overwhelms the true value, the door control unit will be commanded to unlock the doors. When the true value overwhelms the spoofed value, the doors will lock. Thus, the doors will rapidly lock and unlock.

Fig. 9. Schematic diagram of a spoofing attack.

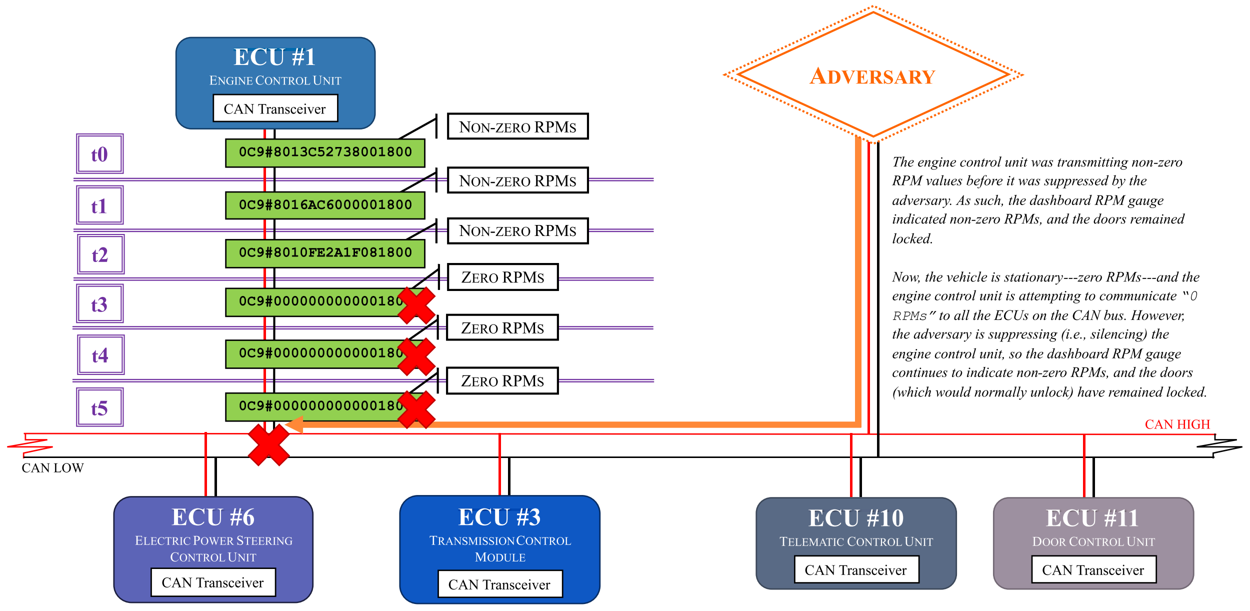
ECU #1
Engine Control Unit
CAN Transceiver
Adversary
t0
0C9#8013C52738001800
Non-zero RPMs
t1
0C9#8016AC6000001800
Non-zero RPMs
t2
0C9#8010FE2A1F081800
Non-zero RPMs
t3
0C9#000000000000180
Zero RPMs
t4
0C9#000000000000180
Zero RPMs
t5
0C9#000000000000180
Zero RPMs
The engine control unit was transmitting non-zero RPM values before it was suppressed by the adversary. As such, the dashboard RPM gauge indicated non-zero RPMs, and the doors remained locked.
Now, the vehicle is stationary---zero RPMs---and the engine control unit is attempting to communicate "0 RPMs" to all the ECUs on the CAN bus. However, the adversary is suppressing (i.e., silencing) the engine control unit, so the dashboard RPM gauge continues to indicate non-zero RPMs, and the doors (which would normally unlock) have remained locked.
CAN HIGH
CAN LOW
ECU #6
Electric Power Steering Control Unit
CAN Transceiver
ECU #3
Transmission Control Module
CAN Transceiver
ECU #10
Telematic Control Unit
CAN Transceiver
ECU #11
Door Control Unit
CAN Transceiver

Fig. 10. Schematic diagram of a suppress attack.

mitting “zero RPMs.” Of course, due to confliction, the needle in the RPM gauge will waffle between “zero RPMs”—the spoofed value—and “high RPMs”—the true value. Similarly, the doors will sporadically lock and unlock.

This type of attack is often a building block for more sophisticated attacks (e.g., masquerade attacks). In a sophisticated masquerade attack, the aforementioned confliction issues—RPM gauge fluctuation, sporadic door locking/unlocking—would be eliminated.

Many of our masquerade/spoofing attacks are replay attacks. During our reconnaissance, we observed CAN frames associated with particular arbitration identifiers and ECUs. We collected CAN messages while the vehicle was in one condition (e.g., the vehicle was traveling at a high speed, the vehicle was in “neutral”), and when the vehicle condition changed (e.g., the vehicle was traveling at low speed, the vehicle was in “reverse”), we replayed the collected messages. As such, we were able to deceive the vehicle into behaving as though it was in “neutral” when it was actually in “reverse.” Similarly, when the vehicle was traveling at low speed, our replay attack manipulated the speedometer into showing a high rate of speed—e.g., 75 mph.

2.2.5. *Spoofing*

Fig. 9 outlines a CAN bus spoofing attack. A spoofing attack is similar to a replay attack, but it goes a step further. The adversary will still need to eavesdrop on CAN communications. To mount a more effectual attack, the adversary will need to determine which CAN frames should be abused in order to achieve the desired outcome (e.g., disruption, damage, death). The adversary will use his or her reconnaissance to craft tailored (i.e., *spoofed*) CAN frames that achieve the desired outcome (Lampe and Meng, 2023a).

As mentioned above, many of our spoofing attacks were also masquerade attacks and replay attacks. See Sections 2.2.3 and 2.2.4 for details.

2.2.6. *Suppress*

Fig. 10 depicts the CAN bus during a suppress attack. This attack stifles (i.e., *suppresses*) communication from a target ECU. In our example, the target ECU is the engine control unit. Oftentimes, a suppress attack will be combined with replay or spoofing in order to conduct a sophisticated masquerade attack; however, a suppress attack can be effective on its own. In Fig. 10, the engine control unit begins by an-

Table 2
Published CAN bus exploits.

Year	Highlights	Description	Source
2010	Manipulating ECUs, embedding malicious code	An analysis of the vulnerabilities of internal vehicle networks that uncovered various types of attacks—e.g., bridging between a vehicle's internal subnetworks, manipulating ECUs (selectively engaging and disabling the brakes), and embedding malicious code in the telematics unit, to name a few.	Koscher et al. (2010)
2014	Diagnostic commands, ECU reprogramming	An analysis of the vulnerabilities of internal vehicle networks that uncovered various types of attacks—e.g., disabling power steering via denial of service attack, sending diagnostic commands (to shut down the engine, to selectively engage and disable the brakes, etc.), and reprogramming ECUs.	Miller and Valasek (2014)
2015	Telematics, remote control	Exploitation of a popular telematic control unit—plugged into the OBD-II port—to achieve remote control of a vehicle.	Foster et al. (2015)
2015	Remote control of an unaltered private automobile	Exploitation of a cellular network vulnerability to achieve arbitrary remote control of an unaltered private automobile—e.g., exfiltrating global positioning system (GPS) coordinates to track the vehicle, locking and unlocking the vehicle, shutting down the engine, etc.	Miller and Valasek (2015a,b)
2016	Advanced CAN injection attacks, steering wheel manipulating at speed	Eliminate confliction in order to conduct advanced CAN injection attacks, such as manipulating the steering wheel at speed.	Miller and Valasek (2016b,a)
2017	Remote control of an unaltered private automobile	A chain of exploits—including browser exploits, local privilege escalation, and ECU reprogramming—was used to compromise and remotely control an unaltered Tesla automobile.	Nie et al. (2017)
2018	Airbag deployment	Penetration testing identified vulnerabilities in a vehicle's pyrotechnic control unit (PCU), which is responsible for airbag deployment. The vulnerabilities were successfully exploited on an automotive testbed, generating the firing impulse that—in a real vehicle—would result in airbag deployment.	Dürrwang et al. (2018)
2019	Remote control of an unaltered private automobile, SMS	Vulnerabilities in a BMW's infotainment and telematics components were used to remotely compromise the vehicle. Remote services were triggered via short message service (SMS) communications, and cross-domain diagnostic commands were used to access additional ECUs.	Cai et al. (2019)
2023	Grand theft auto	Many vehicles provide relatively easy access to the CAN bus from the outside—e.g., via wiring behind a headlight. Thieves can access the CAN bus, inject spoofed CAN frames, suppress conflicting CAN frames, and, ultimately, steal the vehicle.	Tindell (2023)

nouncing non-zero RPMs (presumably, the vehicle is in motion). Later, the engine control unit wishes to communicate that the RPMs have dropped to zero. The door control unit (DCU) knows that the doors should remain locked while the vehicle is in motion—non-zero RPMs—and unlock only when the RPMs are down to zero (this is a simplified example). As such, even though the vehicle is down to zero RPMs at time `t3`, the doors will remain locked. In some vehicles, passengers will not be able to manually override the door locks when the vehicle is in motion—or when the DCU believes the vehicle is in motion.

Our datasets do not contain suppress attacks; we plan to focus on suppress attacks in our future work (See Section 9).

2.3. The exploits

Table 2 highlights a number of published CAN bus exploits, ranging from nuisance attacks to theft to life-threatening remote-controlled attacks.

One of the earliest experimental CAN bus exploits was conducted in 2010 by Koscher et al. (2010). They were able to continuously activate the vehicle's door locks, run the windshield wipers, blare the horn, turn various lights on and off, kill the engine, grind the starter, engage the brakes, disable the brakes, and more.

Miller and Valasek have conducted a number of proof-of-concept CAN bus exploits (Miller and Valasek, 2014, 2015a,b, 2016b,a). Many of the exploits would be life-threatening—even catastrophic—if conducted by malicious adversaries rather than researchers. For example, the attack against a 2014 Jeep Cherokee (Miller and Valasek, 2015a,b) could be generalized to a number of Fiat Chrysler Automotive (FCA) automobiles—including various models from the Chrysler, Dodge, Jeep, and Ram brands. FCA recalled 1.4 million vehicles. If malicious adversaries (e.g., nation-state actors) were to simultaneously compromise those 1.4 million vehicles and disable their brakes, the results could easily be catastrophic.

Miller and Valasek disabled the 2014 Jeep Cherokee's brakes via diagnostic messages, which most of its ECUs would ignore when the vehicle was traveling at speed. Cai et al. (2019), on the other hand, found that most BMW ECUs will respond to diagnostic messages even when the vehicle is driving at normal speeds.

When we zoom out to a "big picture" view, we can see that there are even bigger problems. Innovations such as advanced driver assistance systems (ADAS), automotive platooning, and smart cities are vulnerable to both *intra*-vehicle and *inter*-vehicle attacks. The "smarter" the vehicle, the more devastating the potential attack. If a vehicle lacks electronic steering, then an adversary cannot control the steering. A "smarter" vehicle, equipped with electronic steering (e.g., parking assist, lane-keeping assist), faces the uncomfortable possibility of adversary-controlled steering. Lakhal et al. (2021) identified a number of reliability and security issues related to the modern CAN bus that specifically threaten intelligent transportation systems (ITSs). All in all, we can see that the CAN bus is critical to the future of automotive transportation and innovation.

3. Related work

Constructing a high-fidelity CAN dataset often involves logging—i.e., recording CAN traffic from a real vehicle. Daily and Van (2021) describe a cryptographically-secured CAN logging scheme, which uses encryption and hashing to achieve confidentiality and integrity, respectively.

Marx et al. (2016) analyzed and compared different CAN logging techniques in terms of accuracy and file size. They experimented with both frame- and waveform-based logging strategies. Across all strategies, the mean difference between measurements was less than 0.08%.

Table 3
CAN bus datasets: *A Summary*.

Year	Name	Acronym	Vehicle(s)	# Attacks	Labeled?[a]	Real?	Source
2016	Simulated CAN Bus dataset	Sim CAN	Testbed	1	No	No	Kang and Kang (2016)
2017	CrySyS Lab CAN dataset	CrySyS CAN	Unknown	0	N/A	Yes	CrySyS Lab (2017)
2017	HCRL CAN dataset	HCRL CAN	Kia Soul	3	No	Yes	Lee et al. (2017)
2018	HCRL Car-Hacking dataset	HCRL CH	Hyundai YF Sonata	4	Yes	Yes	Seo et al. (2018); Song et al. (2020)
2018	HCRL Survival Analysis dataset	HCRL SA	Chevrolet Spark, Hyundai YF Sonata, Kia Soul	3	Yes	Yes	Han et al. (2018)
2019	AEGIS Big Data Project Automotive CAN Bus dataset	AEGIS CAN	Unknown	0	N/A	Yes	Kaiser et al. (2019)
2019	Bus-Off dataset	Bus-Off	Volvo V40	2	Yes	Yes[b]	Stabili and Marchetti (2019)
2019	TU Eindhoven CAN bus intrusion dataset v2	TU CAN v2	Opel Astra, Renault Clio, Testbed	5	No	Yes[c]	Dupont et al. (2019)
2020	HCRL CAN Signal Extraction and Translation dataset	HCRL SET	Unknown	0	N/A	Yes	Song and Kim (2020)

The details in the table were aggregated with help from Rajapaksha et al. (2023); Verma et al. (2020, 2022); Karopoulos et al. (2022); Wu et al. (2019); Vahidi et al. (2022), and Lampe and Meng (2023d).

[a] We define a "labeled" dataset as a dataset in which *individual* samples (e.g., *individual* CAN frames) are pre-labeled to indicate whether a particular sample is "normal" or "injected." Therefore, datasets in which metadata is provided to help identify attacks are not counted as labeled because would-be practitioners would still need to label the datasets themselves.

[b] The attack-free data was collected from a real vehicle; the attack data was simulated using the attack-free data.

[c] This dataset contains both real and simulated CAN traffic.

As such, the authors concluded that, accuracy-wise, both frame- and waveform-based logging techniques would be acceptable, though they noted that waveform-based logging consumes less memory and would be desirable for long-term logging.

Reverse engineering is essential to understanding CAN traffic, particularly when devising CAN bus attacks. The quality of a CAN intrusion detection dataset is heavily dependent on the quality of the attacks. Buscemi et al. (2023) surveyed CAN bus reverse engineering methodologies. They reviewed data collection—i.e., CAN logging—as well as the popular `.dbc` file type, which can convert a raw CAN signal into a human-readable, semantic value. Moreover, Buscemi et al. conducted benchmarks of both (1) CAN tokenization algorithms and (2) CAN translation algorithms.

3.1. Existing datasets

There are a number of existing CAN datasets, each with its own advantages and disadvantages. Tables 3 and 4 summarize the existing datasets, providing information such as vehicle type(s), number of attacks, etc. Meanwhile, Tables 5 and 6 furnish details and highlights of the existing datasets, such as attack types, advantages, and limitations.

A common limitation of existing datasets is the lack of attack data; when developing an intrusion detection system (IDS), attack data is of paramount importance. Insufficient data quantity is a second major issue, especially when it comes to machine learning IDSs, as they require a substantial quantity of data for training (and testing). Quality—that is, fidelity—can also be problematic. Researchers are understandably cautious about conducting CAN bus attacks while driving, but some simulations and testbeds are not the best approximations of a real vehicle's CAN bus.

Several works have conducted in-depth reviews of existing datasets. Wu et al. (2019) surveyed intrusion detection for in-vehicle networks and included a discussion of datasets and tools used in previous works.

Karopoulos et al. (2022) developed a meta-taxonomy for IVN-based intrusion detection, in which they included a table of publicly available vehicular intrusion detection—VIDS—datasets. They catalogued VIDS datasets for controller area network as well as Ethernet, Wi-Fi, Bluetooth, and others. Karopoulos et al. enumerated the following features:

1. Presence of train/test sub-datasets
2. Number of features
3. Total number of rows
4. Number of attacks
5. Presence of labels
6. Fidelity (real or simulated)
7. Number of nodes

Rajapaksha et al. (2023) reviewed AI-based intrusion detection systems and included a section on benchmark datasets. They assessed several datasets released by the Hacking and Countermeasure Research Lab (HCRL) (Lee et al., 2017; Seo et al., 2018; Song et al., 2020; Han et al., 2018; Song and Kim, 2020; Kang et al., 2021) as well as a number of others. For each dataset, they sought to identify both advantages and limitations. They concluded that the Real ORNL Automotive Dynamometer CAN intrusion dataset (Verma et al., 2020, 2022) is the most comprehensive CAN intrusion detection dataset available.

Verma et al. (2022) focused specifically on open-access CAN intrusion detection datasets. They exhaustively discuss each dataset, including its (1) quality, (2) benefits, (3) drawbacks, and (4) use case. The authors note that their work is the first comprehensive guide to open-access CAN intrusion detection datasets. In addition, they introduced the ROAD dataset (Verma et al., 2020, 2022), which offers a wide variety of attack captures as well as a high volume of attack-free data. Attack-free data was collected both on a dynamometer—a "rolling road" in which the vehicle remains stationary, but the vehicle operates as though driving normally (de Menezes Lourenço et al., 2022)—and on the open road. All attacks were conducted while the vehicle was on a dynamometer and actively being driven.

4. Methodology

In this section, we discuss our methodology, including our setup, our attack-free data collection, our attack data collection, and our data pre-processing.

We seek to develop a dataset that is suitable for machine learning—both traditional machine learning and deep learning. As such, during data collection, we emphasize (1) quality and (2) quantity. A machine learning IDS depends on a high volume of data for adequate training and testing. The machine learning model will fit to the training data and will be evaluated on the testing data, so if the data is low quality—i.e., low fidelity—then the model will be ill-fitted to the real problem,

Table 4
CAN bus datasets: *A Summary (continued)*.

Year	Name	Acronym	Vehicle(s)	# Attacks	Labeled?[a]	Real?	Source
2020	ML350 CAN Bus dataset	ML350 CAN	Mercedes ML350	2	Yes	Yes	Sami et al. (2020); Sami (2019)
2020	Real ORNL Automotive Dynamometer CAN intrusion dataset	ROAD	Unknown (mid-2010s)	10[b]	No	Yes	Verma et al. (2020); Verma et al. (2022)
2020	Reverse Engineering CAN Bus dataset	ReCAN	Alfa Romeo Giulia Veloce, Opel Corsa, Mitsubishi Fuso Canter, Isuzu M55, Piaggio Porter Maxi	0	N/A	Yes	Zago et al. (2020b,a)
2020	Synthetic CAN Bus dataset	SynCAN	Testbed	5	No	No	Hanselmann et al. (2020)
2021	HCRL Attack & Defense Challenge dataset	HCRL A&D	Hyundai Avante CN7	4	Yes	Yes	Kang et al. (2021)
2021	Heavy-Duty Truck CAN Bus dataset	Truck CAN	Renault T520 6X2	0	N/A	Yes	University of Turku (2021)
2022	Detecting Attacks to in-vehicle networks via n-Gram Analysis dataset	DAGA	Volvo V40	6	Yes	Yes[c]	Stabili et al. (2022)
2023	Ventus dataset	Ventus	Volvo V40	2	Yes	Yes[c]	Pollicino et al. (2023)
2023	`can-train-and-test` dataset[d]	`CT&T`	Chevrolet Impala, Chevrolet Silverado, Chevrolet Traverse, Subaru Forester	9	Yes	Yes[e]	Our contribution

The details in the table were aggregated with help from Rajapaksha et al. (2023); Verma et al. (2020, 2022); Karopoulos et al. (2022); Wu et al. (2019); Vahidi et al. (2022), and Lampe and Meng (2023d).

[a] We define a "labeled" dataset as a dataset in which *individual* samples (e.g., *individual* CAN frames) are pre-labeled to indicate whether a particular sample is "normal" or "injected." Therefore, datasets in which metadata is provided to help identify attacks are not counted as labeled because would-be practitioners would still need to label the datasets themselves.

[b] This dataset contains 33 attack traffic captures; however, the attack types are not unique. There are fuzzing attacks, spoofing & masquerade attacks, and accelerator attacks. Four different signals are spoofed—wheel speed, engine coolant temperature, speedometer value, and reverse light (on/off). For each spoofing attack, Verma et al. provide a masquerade version (legitimate signals are suppressed). We count the fuzzing attack, the four spoofing attacks, the four masquerade attacks, and the accelerator attack as unique attacks, resulting in a total of ten attacks.

[c] The attack-free data was collected from a real vehicle; the attack data was simulated using the attack-free data.

[d] `can-train-and-test` is the name of the repository which contains the final curated dataset (labeled and partitioned). The `can-train-and-test` repository is linked to the `can-dataset` and `can-ml` repositories, which contain raw `.log` files, additional attack-free traffic, and additional attack traffic. As such, we use the term `can-train-and-test` to refer to the `can-train-and-test` repository as well as the linked repositories.

[e] This dataset contains both real and simulated CAN traffic.

and the model's performance during the evaluation will not match its performance when faced with the real problem.

During the data pre-processing phase, we label our dataset so as to facilitate both supervised and unsupervised learning. We provide `.log` files that can be replayed using `SocketCAN` (Community, 2023) and `can-utils` (can-utils Contributors, 2023)—no pre-processing needed—as well as `.csv` files that can be loaded into a program—e.g., the `pandas` (pandas Contributors, 2023) Python library.

4.1. Setup

We collected live, on-the-road data from four different vehicles and six different drivers. Our objective was to capture both vehicle and driver idiosyncrasies in order to construct an authoritative, diversified CAN dataset for machine learning.

Vehicles. For our test vehicles, we varied the manufacturer (Chevrolet, Subaru), the model (Impala, Traverse, Silverado, Forester), and the vehicle type (sedan, sport utility vehicle, pickup truck). Our objective was to collect diverse data, so that a proposed intrusion detection system could be evaluated in terms of its ability to generalize to different vehicles. Our vehicles are enumerated below:

1. *2011 Chevrolet Impala:* A four-door sedan
2. *2011 Chevrolet Traverse:* A full-size sport utility vehicle (SUV)
3. *2016 Chevrolet Silverado:* A four-door pickup truck
4. *2017 Subaru Forester:* A compact SUV

Drivers. For our test drivers, we varied both age and gender, such that our CAN traffic captures would reflect the different driving habits of different people. Our test drivers—in terms of demographics—are enumerated below:

1. Male, under 30 years of age
2. Female, under 30 years of age
3. Male, 30-60 years of age
4. Female, 30-60 years of age
5. Male, over 60 years of age
6. Female, over 60 years of age

Our test drivers are acknowledged at the end of the paper.

Logging. To log CAN traffic data, we accessed each vehicle's onboard diagnostic port (i.e., OBD-II port) (Falch, 2022) (see Fig. 11. U.S. law requires that the OBD-II port be within arm's reach of the driver's seat and accessible without tools (Agency, 1999). Similar legislation exists in the European Union. To interface with the OBD-II port, we selected Korlan USB2CAN, an "8devices" product (8devices, 2023). The cable converts raw OBD-II data into USB data—one end of the cable is plugged directly into the OBD-II port; the opposite end is plugged directly into our laptop. As such, we were able to communicate with the OBD-II port via Linux's `SocketCAN` subsystem (Community, 2023) and `can-utils` utilities (can-utils Contributors, 2023). Our wired connection allowed us to simultaneously inject data and collect data, facilitating real-world injection attacks at speed. We leveraged the Linux command line to send spoofed packets at regular intervals—typically ranging from every 0.1 seconds to every 0.0001 seconds. A few attacks were conducted at even higher or lower intervals. For more complex attacks, we developed a Python program to monitor CAN traffic and send spoofed messages when one or more conditions were met. For exam-

Table 5
CAN bus datasets: *Details*.

Year	Name	Acronym	Details[a]	Source
2016	Simulated CAN Bus dataset	Sim CAN	A 3-ECU testbed was constructed to simulate both benign traffic and generalized injection-type attacks. During the simulation, 200,000 packets were generated using the Open Car Testbed and Network Experiments (OCTANE) (Everett and McCoy, 2013) packet generator.	Kang and Kang (2016)
2017	CrySyS Lab CAN dataset	CrySyS CAN	This dataset consists of purely benign data associated with specific driving scenarios (e.g., "Driving with a speed of 40 km/h, then lane change, then stop"). A "CAN log infector" is provided to tack on attacks in the post-processing stage.	CrySyS Lab (2017)
2017	HCRL CAN dataset	HCRL CAN	This dataset is one of many datasets developed and published by the Hacking and Countermeasure Research Lab (HCRL). It contains benign traffic as well as several types of attacks—DoS, fuzzing, and impersonation (i.e., masquerade). The timestamp, arbitration ID, data length code (DLC), and data field are provided as features. The DoS attack is labeled, but none of the others are.	Lee et al. (2017)
2018	HCRL Car-Hacking dataset	HCRL CH	This dataset contains benign traffic as well as several types of attacks—DoS, fuzzing, RPM spoofing, and gear spoofing. The timestamp, arbitration ID, DLC, data field, and label ("T" for injected, "R" for normal) are provided as features. During data collection, the vehicle was parked with the engine running. According to Rajapaksha et al. (2023), this is the most widely used dataset in the literature (to evaluate CAN IDSs).	Seo et al. (2018); Song et al. (2020)
2018	HCRL Survival Analysis dataset	HCRL SA	Similar to the HCRL Car-Hacking dataset, this dataset contains both benign traffic and attacks—flooding (i.e., DoS), fuzzing, and malfunction. The timestamp, arbitration ID, DLC, data field, and label ("T" for injected, "R" for normal) are provided as features. An attack-free traffic capture is available for each vehicle; however, the traffic captures are not large enough to train a good classifier (Rajapaksha et al., 2023).	Han et al. (2018)
2019	AEGIS Big Data Project Automotive CAN Bus dataset	AEGIS CAN	This dataset consists of purely benign data—specifically, signal data. It is similar to the dataset curated by Hanselmann et al. (2020), though it also contains global positioning system (GPS) data.	Kaiser et al. (2019)
2019	Bus-Off dataset	Bus-Off	This dataset contains two types of simulated bus-off attacks—namely, shutdown and inhibition. In the shutdown attack, messages associated with a particular arbitration ID are removed from the CAN trace for a variable period of time, simulating an attacker who drives an ECU to the "bus-off" state. In the inhibition attack, messages associated with a particular arbitration ID are removed from the entire CAN trace, simulating an attacker who disables an ECU. As such, the two bus-off attacks are *suppress* attacks. There are 189,083,068 total samples in the dataset (8,743,772 attack-free samples and 180,339,296 attack samples).	Stabili and Marchetti (2019)
2019	TU Eindhoven CAN bus intrusion dataset v2	TU CAN v2	This dataset contains benign traffic as well as several types of attacks—DoS, fuzzing, replay, suspension (i.e., suppress), and diagnostic. This dataset contains both real data—from two vehicles—and synthetic data from a testbed. This dataset furnishes the only diagnostic attack publicly available. However, some of the simulated attacks have fidelity shortcomings; for example, the DoS attack was constructed by overwriting ten seconds' worth of traffic—real DoS attacks are much noisier (Verma et al., 2022).	Dupont et al. (2019)
2020	HCRL CAN Signal Extraction and Translation dataset	HCRL SET	This dataset consists of purely benign data; it was designed for signal extraction, not intrusion detection. The timestamp, arbitration ID, DLC, and data field are provided as features.	Song and Kim (2020)

[a] The descriptions in the table were aggregated with help from Rajapaksha et al. (2023); Verma et al. (2020, 2022); Karopoulos et al. (2022); Wu et al. (2019); Vahidi et al. (2022), and Lampe and Meng (2023d).

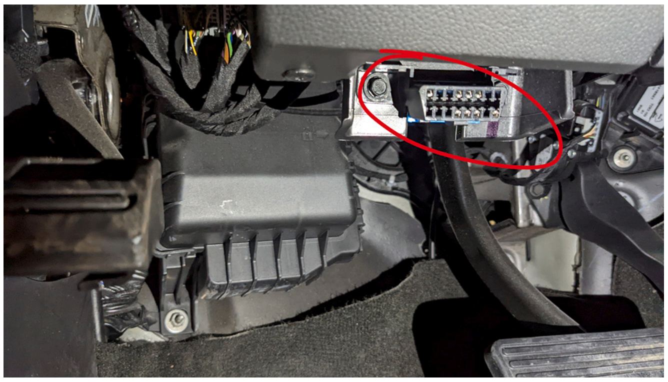
Fig. 11. The OBD-II port (Falch, 2022) of a 2016 Chevrolet Silverado. We accessed this port—via Korlan USB2CAN (8devices, 2023)—in order to capture CAN traffic and conduct attacks.

ple, our "interval" attack (see Section 4.3) injects one or more messages whenever a particular arbitration ID crosses the CAN bus.

Our test driver sat in the driver's seat, while our experimenter sat in the front passenger seat, holding the laptop and managing the data collection process.

Data. For each CAN frame, we captured the `timestamp`, `arbitration identifier`, and `data field`. The remaining fields—`SOF`, `DLC`, `CRC`, etc.—can be reconstructed using the captured fields and the CAN frame specifications. For example, the `DLC` is simply the length of the `data field`. Moreover, to replay the captured CAN frames using the `canplayer` command from `can-utils` (see Section 6.1), we only need the `timestamp`, `arbitration identifier`, and `data field`.

4.2. *Attack-free data*

We captured attack-free CAN traffic under two conditions:

1. Driving mode
2. Accessory mode

While serious (i.e., life-threatening) automotive attacks are generally associated with driving conditions, a number of disconcerting automotive attacks are better suited to stationary settings. For example, an automobile could be tracked while parked. If the automobile connects to home Wi-Fi when parked in the garage, then an attacker could exfiltrate data while the automobile is parked and connected. In addition, Tindell discussed in detail how thieves can access the CAN bus of a parked vehicle in order to steal it (Tindell, 2023) (see Table 2).

Therefore, we collected attack-free traffic in both driving and non-driving settings. With our two types of attack-free data, an IDS can be trained to develop "normal" baselines for both driving and accessory modes. Since automotive IDSs are typically anomaly-based,

Table 6
CAN bus datasets: *Details* (continued).

Year	Name	Acronym	Details[a]	Source
2020	ML350 CAN Bus dataset	ML350 CAN	This dataset was collected from a real vehicle using a CL2000 CAN bus logger. It contains both benign traffic and attacks—DoS and fuzzing. The label “R” refers to attack-free messages, while the label “T” identifies attack messages.	Sami et al. (2020); Sami (2019)
2020	Real ORNL Automotive Dynamometer CAN intrusion dataset	ROAD	This dataset contains benign traffic as well as several types of attacks—fuzzing, targeted ID attacks (i.e., spoofing, masquerade), and accelerator attacks. The attacks (except the masquerade attacks) were conducted while the test vehicle was on a dynamometer (a “rolling road”—the vehicle remains stationary, but the wheels turn as though driving normally (de Menezes Lourenço et al., 2022)); as such, the attacks are high fidelity. The timestamp, arbitration ID, and data field are provided as features; the dataset is unlabeled.	Verma et al. (2020); Verma et al. (2022)
2020	Reverse Engineering CAN Bus dataset	ReCAN	This dataset contains purely benign data from both passenger vehicles and commercial trucks, and it includes both raw data and decoded signal data. The dataset was developed to support reverse engineering of controller area networks.	Zago et al. (2020b,a)
2020	Synthetic CAN Bus dataset	SynCAN	This dataset contains benign traffic as well as several types of attacks—flooding (i.e., DoS), playback (i.e., replay), suppress, plateau, and continuous. However, it contains only signal values—no raw CAN data. In this regard, it is similar to the dataset curated by Kaiser et al. (2019). All attacks are simulated; as such, the real impacts—if any—on real vehicles are not known. According to Rajapaksha et al. (2023), this is the most widely used dataset to evaluate unsupervised payload-based CAN IDSs.	Hanselmann et al. (2020)
2021	HCRL Attack & Defense Challenge dataset	HCRL A&D	This dataset contains benign traffic as well as several types of attacks—flooding (i.e., DoS), fuzzing, replay, and spoofing. The timestamp, arbitration ID, DLC, data field, class label (normal or attack), and subclass label (attack type) are provided as features. Generally, the attacks were conducted when the vehicle was stationary—due to safety concerns.	Kang et al. (2021)
2021	Heavy-Duty Truck CAN Bus dataset	Truck CAN	This dataset furnishes benign data which conforms to the SAE J1939 standard. The data was collected from a heavy-duty truck and is contained in comma-separated values (CSV) files. Each file is composed of 50,000 CAN frames—approximately one minute’s worth of CAN traffic.	University of Turku (2021)
2022	Detecting Attacks to in-vehicle networks via n-Gram Analysis dataset	DAGA	This dataset contains six attacks, each of which was simulated using data originally collected from an unaltered, licensed passenger vehicle. The six attacks include replay attacks (single arbitration ID replay, arbitrary sequence replay, ordered sequence replay), fuzzing attacks (message arbitration ID fuzzing, payload fuzzing), and a denial of service (DoS) attack. There are 256,301,010 total samples in the dataset (8,743,772 attack-free samples and 247,557,238 attack samples).	Stabili et al. (2022)
2023	Ventus dataset	Ventus	This dataset consists of injection and removal (i.e., suppress) attacks. There are 539,657,925 total samples in the dataset (8,743,772 attack-free samples and 531,003,538 attack samples).	Pollicino et al. (2023)
2023	`can-train-and-test` dataset[2]	CT&T	This dataset contains several simulated attacks as well as live attacks on real vehicles (the attacks were conducted when the vehicle was being driven down rural roads). Both labeled and unlabeled data is provided; for the labeled data, “0” denotes an attack-free CAN frame, while “1” identifies an attack frame. The attack type is specified by the file name. The curated dataset—`can-train-and-test`—enables researchers to assess the capability of a proposed intrusion detection system under several conditions (e.g., known vehicle, known attack; unknown vehicle, known attack; etc.).	Our contribution

[a] The descriptions in the table were aggregated with help from Rajapaksha et al. (2023); Verma et al. (2020, 2022); Karopoulos et al. (2022); Wu et al. (2019); Vahidi et al. (2022), and Lampe and Meng (2023d).

high-quality baselines are paramount. Without high-quality baselines, anomaly-based IDSs would be hard-pressed to distinguish between “normal” traffic and anomalous traffic.

If an automotive IDS struggles to differentiate attack-free traffic and attack traffic, then it will report a high volume of false positives. False positives are problematic because they dilute the power of a real warning in the event of an actual attack. If drivers are accustomed to false alarms, they may not heed the real warning. If the false alarms are especially frequent and especially irritating, then they may disable the intrusion detection system, leaving the drivers, their passengers, and their vehicles unprotected.

Since driving mode and accessory mode involve markedly different CAN traffic patterns, we found it prudent to include both types of traffic captures. In accessory mode, many ECUs are either asleep—not running—or silent—not communicating—because they are extraneous when the engine is off and the vehicle is stationary. There are fewer arbitration identifiers and far fewer CAN frames per unit of time. If an IDS were trained exclusively in driving mode and implemented in a vehicle that is occasionally operated in accessory mode, then the loss of many arbitration IDs (and many CAN frames) would probably result in a barrage of false positives. As such, we find it prudent to include both driving and non-driving traffic captures, so that an IDS can be trained for both scenarios.

Listings 1 and 2 showcase attack-free traffic in driving mode and accessory mode, respectively. The traffic was collected from the 2011 Chevrolet Traverse. Extended attack-free traces can be found in the appendix (see Appendix A.1).

Listing 1 Attack-free traffic (driving mode).

```
(1672163108.460031) can0 1E1#0000100000
(1672163108.460701) can0 0C7#00D4A352
(1672163108.460932) can0 0F9#031A4002A23FDEFA
(1672163108.461181) can0 189#4FFF0FFF3000DEFA
(1672163108.461405) can0 199#4FFF0E70F18F00FF
(1672163108.461548) can0 1EB#0141
(1672163108.462257) can0 1F1#AE0E00000800007A
(1672163108.462765) can0 0C1#236F7087236A5491
(1672163108.462912) can0 1CB#10E600
(1672163108.463124) can0 0C5#234188A42341E475
(1672163108.463359) can0 184#0001000001FF
(1672163108.463584) can0 1C7#0FFF700003FF3F
(1672163108.463758) can0 1CD#C7FF07FE7F
(1672163108.464028) can0 1E5#4600D8EED4FF2500
(1672163108.464229) can0 1E9#0FFF000C00260000
(1672163108.464477) can0 0C9#801D551911000000
(1672163108.464634) can0 0F1#1C040040
(1672163108.464846) can0 191#075E075E0761115D
```

Listing 2 Attack free traffic (accessory mode).

```
(1672176421.034132) can0 3C1#0765040000000000
(1672176421.034383) can0 3D1#1046000000000000
(1672176421.034632) can0 3E9#0000000000000000
(1672176421.035391) can0 0F1#1C050040
(1672176421.035959) can0 0C7#03FE0000
(1672176421.036231) can0 0F9#00004000000000FF
(1672176421.036505) can0 189#CFFF0FFF2FFE00FF
(1672176421.036735) can0 199#CFFF0E70F18D00FF
(1672176421.037429) can0 1EB#008A
(1672176421.038562) can0 1F3#C0E0
(1672176421.040835) can0 0C1#1000000010000000
(1672176421.041107) can0 0C5#1000000010000000
(1672176421.041350) can0 0C9#0000000D00000000
(1672176421.041577) can0 191#063D091E091E0000
(1672176421.041812) can0 1E5#46FFC2C000003C00
(1672176421.042055) can0 1ED#0000000000000800
(1672176421.042300) can0 1A1#0010010000000000
(1672176421.042552) can0 1C3#063D063D00000000
```

4.3. Attack data

We conducted and recorded the following types of attacks:

1. Denial of Service (DoS)
2. Combined spoofing
   (a) Double spoofing
   (b) Triple spoofing
3. Fuzzing
4. Gear spoofing
5. Interval
6. RPM spoofing
   (a) Driving mode
   (b) Accessory mode
7. Speed spoofing
   (a) Driving mode
   (b) Accessory mode
8. Standstill
9. Systematic

4.3.1. Denial of service

In controller area networks, the highest-priority identifier is 000. As such, when two ECUs begin transmitting at the same time, 000 will win the arbitration—in principle. Often, a new or high-end vehicle will be equipped with a protected CAN bus. A node transmitting invalid arbitration IDs will be kicked off the bus. In the Chevrolet Impala and Traverse, the highest-priority ***valid*** arbitration identifier (that we observed) is 0C1. In the Chevrolet Silverado, it is 0AA, and in the Subaru Forester, it is 002. In the Chevrolet Silverado, we provide a trace of a painfully obvious DoS attack—arbitration ID 000 and data field 0000000000000000. Then, we include a trace in which the highest-priority valid arbitration ID is used—arbitration ID 0AA—but the data field is still all zeroes. Lastly, we conduct a DoS attack with the highest-priority valid arbitration ID, 0AA, and a non-zero data field, 2F042D9002526F00. We conduct similar attacks against the Impala, Traverse, and Subaru.

Listings 3 and 4 both showcase Denial of Service (DoS) attacks. In Listing 3, the DoS attack is extremely overt (the invalid arbitration ID 000 is used), whereas in Listing 4, the DoS attack is more subtle (the valid arbitration ID 0C1 is used). The traffic was collected from the 2011 Chevrolet Traverse. Extended DoS traces can be found in the appendix (see Appendix A.2).

4.3.2. Fuzzing

Traditional fuzzing attacks inject unexpected, invalid, or random data to conduct reconnaissance and pinpoint vulnerabilities. In our fuzzing attacks, we randomize the arbitration identifier, the data field, and the length of the data. We include both valid and invalid arbitration identifiers. With our fuzzing attacks, we do not violate CAN frame specifications, as messages that do not conform to specifications can be trivially rejected by the CAN bus (and, in general, they are). For the Subaru Forester, one fuzzed CAN frame is 58A#9F2F306EBAE7D66A, one is 0D1#3C050080 and one is 26C#3E.

Given that fuzzing attacks involve a high volume of invalid arbitration identifiers, one might assume that they are trivial to detect. A review of the current literature indicates that the opposite is true: because fuzzing attacks are random, many intrusion detection systems—especially machine learning IDSs—struggle to recognize them. Even a high-performing IDS will generally demonstrate a slight drop in detection accuracy when pitted against fuzzing attack (Lampe and Meng, 2023b).

Listing 5 demonstrates a fuzzing attack. The traffic was collected from the 2011 Chevrolet Traverse. Extended fuzzing traces can be found in the appendix (see Appendix A.3).

Listing 3 An overt Denial of Service (DoS) attack. The red lines indicate attack CAN frames. (For interpretation of the colors in the listings, the reader is referred to the web version of this article.)

```
(1672451549.638534) can0 1E9#0006000C00000000
(1672451549.638538) can0 334#0000
(1672451549.638667) can0 000#0000000000000000
(1672451549.639445) can0 000#0000000000000000
(1672451549.639596) can0 1EB#0141
(1672451549.640116) can0 000#0000000000000000
(1672451549.641030) can0 000#0000000000000000
(1672451549.641747) can0 0C9#840BD30700000000
(1672451549.641756) can0 191#06CD06DC06CE0000
(1672451549.641758) can0 1ED#413E018F01820800
(1672451549.641760) can0 1EF#00000215
(1672451549.642163) can0 000#0000000000000000
(1672451549.642824) can0 1A1#0000414000000000
(1672451549.642833) can0 1C3#06CD06CE00000000
(1672451549.643005) can0 000#0000000000000000
(1672451549.644113) can0 000#0000000000000000
(1672451549.644962) can0 19D#40003FFF0002C58F
(1672451549.644971) can0 1AF#000000
```

Listing 4 A subtle Denial of Service (DoS) attack. The red lines indicate attack CAN frames.

```
(1672452653.717352) can0 2C3#084306D006D04600
(1672452653.717356) can0 0F1#3E5100C0
(1672452653.718133) can0 0C1#23F612A62309F450
(1672452653.718908) can0 0C1#23F612A62309F450
(1672452653.719477) can0 0C7#020AC94F
(1672452653.719481) can0 0F9#07CD4000001D60FF
(1672452653.719483) can0 189#CFFF0FFF2FFE60FF
(1672452653.719485) can0 199#CFFF0E70F18D00FF
(1672452653.719723) can0 0C1#23F612A62309F450
(1672452653.720350) can0 0C1#23F612A62309F450
(1672452653.721356) can0 0C1#23F612A62309F450
(1672452653.722361) can0 0C1#23F612A62309F450
(1672452653.722675) can0 1EB#011C
(1672452653.722980) can0 0C1#23F612A62309F450
(1672452653.723500) can0 0C1#23F612A62309F450
(1672452653.724011) can0 0C1#23F612A62309F450
(1672452653.724863) can0 0C9#840B010A00010000
(1672452653.724869) can0 191#06B506C306B50000
```

4.3.3. Spoofing

For our RPM spoofing and speed spoofing attacks, we spoofed low, high, and zero values—e.g., 10 miles per hour (mph), 60 mph, and 0 mph for speed spoofing. In addition, we conducted both overt and

Listing 5 A fuzzing attack. The red lines indicate attack CAN frames.

```
(1672452917.316296) can0 348#00000000
(1672452917.316301) can0 34A#00000000
(1672452917.317073) can0 036#F237
(1672452917.318151) can0 0EE#750F037DB0973413
(1672452917.318595) can0 0F1#34050040
(1672452917.319220) can0 63C#BB98131FFB6BB64D
(1672452917.319747) can0 19D#C0003FFD000000FF
(1672452917.319763) can0 1AF#000000
(1672452917.320301) can0 758#B4E4B84DB744242D
(1672452917.320835) can0 1F5#0F0F002100000300
(1672452917.320848) can0 1EB#0000
(1672452917.321445) can0 1B9#619E732063822815
(1672452917.322642) can0 107#A60ED87316918238
(1672452917.323068) can0 0C1#0000000000000000
(1672452917.323789) can0 3AA#9442CD7C6E7A727F
(1672452917.324170) can0 0C5#000000000002A384
(1672452917.324188) can0 184#0002000001FE
(1672452917.324193) can0 0C9#0000000A00000000
```

subtle spoofing attacks. An overt attack, for example, might spoof 60 mph while the vehicle is actually traveling at a sedate 10 mph. In our more subtle attacks, we would spoof something along the lines of 61 or 62 mph while the vehicle was actually traveling at 60 mph. For our gear spoofing attacks, we spoofed “neutral” (“N”) to avoid damaging the vehicle—in particular, the transmission. We conducted successful “neutral” gear spoofing attacks both when the vehicle was actually in “drive” (“D”) and when the vehicle was actually in “reverse” (“R”). We believe similar attacks are possible with different combinations of gears; however, if we were to successfully spoof “reverse” while the vehicle was in “drive,” we would almost certainly damage the transmission (i.e., strip the gears). Such an attack would also be dangerous, especially at speed.

In the Chevrolet Silverado, the arbitration ID `135` announces the vehicle’s current gear (and a lot more signals, which are not well understood). The first byte—the most significant byte—signifies the Silverado’s current gear. In the CAN frame `135#0008124ACC161601`, the most significant byte—`00`—says that the vehicle is in “park.” In the CAN frame `135#0100175ECC161619`, byte `01` gives the instruction for “neutral,” and in `135#0200175ECC16160E`, byte `02` gives the instruction for “drive.” If the most significant byte were `03`, the instruction would be “reverse.” The arbitration ID responsible for the transmission varies by manufacturer—and sometimes even by model. For the Chevrolet Impala and Chevrolet Traverse, the arbitration ID `1F5` is used. For the Subaru Forester, it is `148`.

During our double- and triple-spoofing attacks, we spoofed multiple signals (e.g., gear, RPMs, speed) simultaneously. When we conducted our combined spoofing attacks against real automobiles, we found that the same vehicle would tolerate different frequencies for different signals. E.g., the Traverse and Silverado would tolerate higher frequencies for speed-spoofing attacks than for RPM spoofing attacks. We believe that the speed value is used exclusively for the dashboard speedometer—relatively unimportant—while the RPM value influences automatic shifting, locking and unlocking the doors, etc. Since the RPM value more greatly impacts the vehicle, it is not surprising that the tolerance for suspicious RPM communications is lower.

Listings 6 and 7 showcase spoofing attacks—a gear spoofing attack and a triple spoofing attack, respectively. In the gear spoofing attack, the “neutral” gear is spoofed; the true value is “drive.” In the triple spoofing attack, the gear, the RPMs, and the speed are all spoofed (arbitration IDs `1F5`, `0C9`, `3E9`, respectively). The vehicle is driving at speed, but the spoofed messages indicate that it is in “neutral” with no acceleration and negligible speed. The traffic was collected from the 2011 Chevrolet Traverse. Extended spoofing traces can be found in the appendix (see Appendix A.4).

Listing 6 A gear spoofing attack. The red lines indicate attack CAN frames.

```
(1674135810.232272) can0 34A#08D108CA
(1674135810.234101) can0 1F5#0D0D000300000300
(1674135810.235464) can0 1EB#0144
(1674135810.236563) can0 0C9#8025E6193B000000
(1674135810.236578) can0 191#07C407C407C43BAF
(1674135810.236583) can0 1ED#412F0B970C140800
(1674135810.237649) can0 1EF#00001119
(1674135810.237659) can0 2C3#0949066406648100
(1674135810.237666) can0 0C7#01B4671C
(1674135810.237676) can0 0F9#016E4007A457BB12
(1674135810.238748) can0 189#8FFF0FFF2FFFBB12
(1674135810.238758) can0 0F1#00060040
(1674135810.238767) can0 199#8FFF0E70F18E00FF
(1674135810.239824) can0 0C1#11DBBB58123AAF44
(1674135810.239829) can0 0C5#11805719120CB19E
(1674135810.239834) can0 1E5#46FFE0C000001E00
(1674135810.245141) can0 1CB#10E800
(1674135810.245806) can0 1F5#0D0D000300000300
```

Listing 7 A triple spoofing attack. The red lines indicate attack CAN frames.

```
(1674122022.805533) can0 1ED#412E019C01EB0733
(1674122022.805541) can0 1EF#0000023C
(1674122022.805548) can0 1A1#0000414000000000
(1674122022.805575) can0 1F5#0D0D000300000800
(1674122022.806616) can0 1C3#06C606C700000000
(1674122022.807686) can0 0C1#1295096812C24242
(1674122022.807696) can0 0C5#12524E0412911683
(1674122022.807703) can0 1E5#46078EC514F87000
(1674122022.807710) can0 0F1#28070040
(1674122022.808097) can0 3E9#0000000100000000
(1674122022.808774) can0 1E1#0000100000
(1674122022.808852) can0 0C9#00000000000018
(1674122022.809965) can0 1F5#0D0D000300000800
(1674122022.812655) can0 3E9#0000000100000000
(1674122022.813617) can0 0C9#00000000000018
(1674122022.814539) can0 1F5#0D0D000300000800
(1674122022.815117) can0 0C9#840B560A00000000
(1674122022.815122) can0 0C7#004791A8
```

4.3.4. Interval

Our interval attack leverages a Python script that interfaces with `SocketCAN` (Community, 2023) via the `python-can` package (python-can Contributors, 2023b,a). The script watches the CAN bus for a pre-determined target arbitration ID. When the script recognizes the target arbitration ID, it sends one or more spoofed CAN frames with the same arbitration ID. We varied the number of spoofed CAN frames transmitted by the Python script—for some attacks, we sent just one spoofed CAN frame; for others, we sent several spoofed CAN frames. If timed correctly, this type of attack can spoof a vehicle’s gear, RPMs, speed, etc. with a minimal number of injected CAN frames. For example, if we send just one spoofed speed message upon receipt of a legitimate speed message, we can keep the speedometer needle hovering near the spoofed value. If, instead, we send spoofed speed messages according to a preset frequency, then we need to send many, many more messages to achieve the same effect. Here is an example from the Chevrolet Impala: the arbitration ID `3E9` is associated with the vehicle’s speed. Several arbitration IDs are associated with speed (e.g., the speeds of individual wheels), but `3E9` contains the speed signal that is displayed on the speedometer. Our Python script sees the CAN frame `3E9#05CB087005BA0849`, a non-zero speed. Immediately, our script communicates `3E9#0000000000000000` (zero speed). The needle inside the speedometer will rise slightly upon receipt of the true speed reading, but before it can reach the vehicle’s actual speed, it will drop back to zero. With just one injected message per true message, we can ensure that the displayed speed is hovering around zero, even though the vehicle is actually traveling much faster. Since the frequency change for the arbitration ID `3E9` is minimal (exactly double the normal frequency), it is much harder to detect than a spoofing attack that relies on a high volume of spoofed messages to overwhelm the true messages.

Listing 8 demonstrates an interval attack, in which the target arbitration ID appears on the CAN bus, and, immediately, the adversary transmits a spoofed message to countermand the true message. Depending on bus traffic and arbitration, the spoofed message may directly follow the true message, or there may be a few messages in between. The traffic was collected from the 2011 Chevrolet Traverse. Extended interval attack traces can be found in the appendix (see Appendix A.5).

Listing 8 An interval attack. The red lines indicate attack CAN frames.

```
(1674033486.438066) can0 1CD#C7FF07FE7F
(1674033486.438074) can0 1E5#460081E000FF7C00
(1674033486.438079) can0 1E9#4FFB000C00000000
(1674033486.438933) can0 1E9#000A000C00060000
(1674033486.440201) can0 1CB#100000
(1674033486.440212) can0 2D1#030000000000
(1674033486.441283) can0 3FD#003A3A
(1674033486.441294) can0 0F1#7E7F00C0
(1674033486.442361) can0 0C9#840AE80A00010000
...
(1674033486.457486) can0 1C7#0FFFAFFF03FF3F
(1674033486.457491) can0 1CD#07FF08017F
(1674033486.458560) can0 1E5#460081A000FF7E00
(1674033486.458575) can0 1E9#4FFA000C00000000
(1674033486.458596) can0 1EB#019B
(1674033486.459399) can0 1E9#000A000C00060000
(1674033486.460715) can0 1CB#100000
(1674033486.460729) can0 0C7#03FE0000
```

4.3.5. *Standstill*

Our standstill attack is a spoofing attack, albeit a peculiar one. The effects of our standstill attack are dependent upon the vehicle as well as the current driving conditions. If the attack is conducted in the 2011 Chevrolet Traverse or the 2016 Chevrolet Silverado under the correct conditions, then it will put the vehicle in “standstill” mode. Essentially, the vehicle behaves as though in neutral: pressing on the accelerator will rev the engine, but the vehicle will not accelerate. The arbitration ID associated with “standstill” mode is also the arbitration ID associated with RPMs. To trigger “standstill” mode, an attacker must also spoof zero (or near-zero) RPMs. If there is significant confliction, the attack will fail. This spoofing attack is noteworthy because, normally, in the 2016 Chevrolet Silverado, our “malicious” node will be ejected from the CAN bus within a few seconds if it sends spoofed CAN frames at even a moderate frequency. However, when spoofing “standstill” mode, we were able to inject messages at an extremely high frequency without getting kicked off the CAN bus. In the Chevrolet Traverse, we injected the CAN frame `0C9#0000000000004008` to trigger “standstill” mode. In the Chevrolet Silverado, we injected `0C9#0000000700010000` to achieve an equivalent effect.

Listing 9 outlines a standstill attack, in which the vehicle is tricked into behaving as though in “neutral.” The traffic was collected from the 2011 Chevrolet Traverse. Extended standstill attack traces can be found in the appendix (see Appendix A.6).

4.3.6. *Systematic*

Our systematic attack is similar to our fuzzing attack; however, we do not generate the arbitration identifiers at random. Instead, we generate them *systematically*. As such, we can iterate over all possible 11-bit arbitration IDs. Logically, an attacker might conduct such an attack to determine which arbitration IDs are valid. Similarly, an attacker might use a systematic approach to assess the impact of each arbitration ID. From an IDS's perspective, a systematic attack will look markedly different from a fuzzing attack. Therefore, we feel it is important to include such an attack in order to adequately train and test a machine learning IDS. The following are a sequence of CAN frames transmitted during a systematic attack against the Subaru Forester: `000#06B01E39C90F4A15`, `001#B1E7C157D72E`, `002#39B460112D4687`, and `003#E8200020EE`. The arbitration ID increases incrementally with each new message, and the data field is randomized to obfuscate the attack.

Listing 10 spotlights a systematic attack, which might be used for reconnaissance. The traffic was collected from the 2011 Chevrolet Traverse. Extended systematic attack traces can be found in the appendix (see Appendix A.7).

Listing 9 A standstill attack. The red lines indicate attack CAN frames.

```
(1674135909.661269) can0 1EF#00000252
(1674135909.661280) can0 3C9#0766000000000000
(1674135909.661382) can0 0C9#0000000000004008
(1674135909.662355) can0 1EB#0151
(1674135909.662365) can0 0C7#02A02F6D
(1674135909.663433) can0 0F9#01004006A7ACCF1C
(1674135909.663444) can0 0F1#3E4300C0
...
(1674135909.671051) can0 1ED#4144000001AC0860
(1674135909.671061) can0 1EF#00000255
(1674135909.671069) can0 1A1#0000414000000000
(1674135909.671079) can0 1C3#0634063500000000
(1674135909.672144) can0 2C3#0897063506353600
(1674135909.673221) can0 0F1#0A4300C0
(1674135909.673231) can0 0C1#334C4B0F33079579
(1674135909.673256) can0 0C9#0000000000004008
(1674135909.674302) can0 0C5#32C6FDDC32BCA5B3
(1674135909.674314) can0 1E5#46FFF88000000800
```

Listing 10 A systematic attack. The red lines indicate attack CAN frames.

```
(1672454739.897283) can0 1E5#46FFEFE000000E00
(1672454739.898351) can0 1E9#001D000C00030000
(1672454739.899311) can0 000#98BF3D7C9A5CC42F
(1672454739.899415) can0 0C7#018CBBB7
(1672454739.900398) can0 001#207215535845D22C
(1672454739.900485) can0 0F9#022340041C15FC0D
(1672454739.900495) can0 189#0FFF0FFF3001FC0D
(1672454739.900505) can0 199#0FFF0E70F19000FF
(1672454739.901544) can0 002#9C8B734717464C1F
(1672454739.901571) can0 0C9#8020C91337000000
(1672454739.901579) can0 191#07E807E807E837BC
(1672454739.901588) can0 1ED#412F0CB50B9D0800
(1672454739.901592) can0 1EF#000010C3
(1672454739.902660) can0 3F9#0012573257695414
(1672454739.902670) can0 1EB#011D
(1672454739.902680) can0 3FB#8100
(1672454739.902688) can0 003#3473696411F5
(1672454739.903746) can0 1CB#101E00
```

4.4. *Novelty & fidelity*

Novelty. For two of our attacks—RPM spoofing and speed spoofing—we included both a driving mode and an accessory mode variant. As mentioned earlier, there are stark differences between driving and non-driving CAN traffic captures, and it is important to properly train an IDS for both scenarios.

The denial of service (DoS), fuzzing, and spoofing attack families are common in the literature. However, few existent CAN datasets provide multiple spoofing attacks. Moreover, to our knowledge, none of the existing open-access CAN datasets contain combined spoofing attacks—i.e., spoofing attacks that involve spoofing multiple signals simultaneously. In a double-spoofing attack, for example, an adversary might spoof both the vehicle's gear and RPMs. In some automobiles, the ECU(s) associated with gear shifting will ignore shifting commands when the vehicle's RPMs are non-zero. To conduct a gear shifting attack, an adversary might need to spoof both the vehicle's gear and RPMs simultaneously.

Many CAN datasets do not—to our knowledge—account for the defenses that automotive manufacturers have begun to incorporate into CAN buses and ECUs. In our spoofing attacks, we encountered such defenses and crafted attacks capable of eluding them. For example, in newer vehicles, our "malicious" node would be kicked off the bus if we sent too many spoofed messages at too high a frequency. For our attack to succeed, we needed to send enough CAN frames to drown out the legitimate messages coming from the legitimate ECU, but, at the same time, we needed to remain under the threshold that would eject our "malicious" node from the CAN bus. We experimented with various CAN packet frequencies to optimize our attacks. With the RPM spoofing attacks, for instance, we selected a frequency that would point the tachometer needle to the spoofed value, minimizing fluctuation toward the true value without precipitating our ejection from the CAN bus. The optimal frequency differed by vehicle—the 2011 Chevrolet Impala tolerated extremely high frequencies without kicking us off the CAN bus; the 2016 Chevrolet Silverado would exclude our "malicious" node from CAN communications within a couple of seconds unless we were very conservative with our frequencies.

Fidelity. Generally, the spoofing attacks—gear, RPM, and speed—as well as the standstill attack were injected into a real automobile under driving conditions. To alleviate safety concerns, the attacks were first conducted while the vehicle was parked in accessory mode. We selected attacks with visible—but harmless—impacts. Then, the attacks were conducted again, this time at speed in a rural area. The speed spoofing attack impacts only the speedometer. In some vehicles, the RPM spoofing attack impacts only the tachometer; in others, it alters the vehicle's automatic shifting (i.e., it confuses the computer that controls automatic shifting). The gear spoofing and standstill attacks can cause the vehicle to go into neutral or behave as though in neutral. The test drivers were prepared for the attacks, and the attacker was prepared to stop the attack if an unexpected issue occurred.

Out of concern about safety issues or possible damage to the vehicle, the DoS, fuzzing, systematic, and interval attacks were simulated—the attack-free traffic was replayed using Linux's `SocketCAN` subsystem (Community, 2023) and `can-utils` utilities (can-utils Contributors, 2023), the attack traffic was injected using Python, and the resulting traffic was captured.

We conducted real-world attacks against the Chevrolet Impala, Traverse, and Silverado. Due to the owners' concerns, we did not attack the Subaru Forester. All attacks involving the Subaru Forester were simulated.

For a few of our spoofing and standstill attacks, we determined that it would not be feasible to label our captured attack traffic—that is, the attack traffic captured during real attacks on real automobiles. As such, we preserved the real-world traffic captures in our unlabeled datasets, but, for our labeled datasets, we instead conducted simulated attacks that matched the real attacks as closely as possible.

4.5. Pre-processing

We provide our raw dataset, `can-dataset`; a pre-processed and labeled dataset, `can-ml`; and a curated training and testing dataset, `can-train-and-test`. Developing `can-ml` and `can-train-and-test` necessitated a number of pre-processing steps (`can-train-and-test` was curated from `can-ml`). In addition, for all our datasets, we conducted several validation checks to ensure the quality of our data. In this section, we outline six of our pre-processing steps, namely:

1. `.log` file validation
2. `.log` file standardization
3. `.log` file conversion (to `.csv`)
4. `.csv` file standardization
5. `.csv` file labeling
6. Sub-dataset curation

We leveraged the Python programming language to effectuate the bulk of our pre-processing efforts. In particular, we utilized the `python-can` package (python-can Contributors, 2023b,a), which facilitates controller area networking in Python, as well as the `pandas` package (pandas Contributors, 2023), which facilitates data analysis and manipulation.

Validating the `.log` files. We should be able to replay a valid `.log` file using `canplayer` (one of the Linux `can-utils` utilities (can-utils Contributors, 2023)). Playback should be error-free. To validate our `.log` files, we replayed each one and confirmed that it ran without error.

Standardizing the `.log` files. We set up multiple CAN interfaces—both physical and virtual—for our CAN traffic captures. A physical CAN interface, for example, might have been named `can0`, `can1`, etc.; similarly, a virtual CAN interface might have been named `vcan0`, `vcan1`, etc. During data collection and attack generation, we used multiple interfaces in order to conduct multiple experiments simultaneously. For ease of use, we adopt `can0` as our standard CAN interface; that is, we replace all CAN interfaces in all `.log` files with `can0`. It is much easier to use `canplayer` to replay CAN `.log` files when the CAN interface is uniform. In addition, because the CAN interface does not constitute a meaningful feature, it could prove problematic for a machine learning IDS—i.e., the CAN interface could pollute the IDS with noise. By standardizing the CAN interface, we mitigate this issue.

Converting the `.log` files to `.csv` files. Next, we convert the `.log` files to `.csv` files. We preserve only meaningful features—i.e., the timestamp, arbitration identifier, and data field (the CAN interface is discarded). During the conversion process, the CAN frame (e.g., `191#0670069E067E0000`) is split into two separate features: arbitration ID (e.g., `191`) and data field (e.g., `0670069E067E0000`).

Standardizing the `.csv` files. A machine learning IDS can fit to arbitrary, ineffectual patterns in a particular dataset's timestamps. For example, if attack-free data was collected on September 12th, and attack data was collected on September 13th, then a machine learning IDS might distinguish attack data from attack-free data by checking the timestamp. This behavior is counterproductive; the IDS is fitting to noise in the data. Therefore, we standardized all of our `.csv` files to January 1st, 2022. For a given `.csv` file, the timestamp of the first CAN frame is set to January 1st, and the difference between the original timestamp and the January 1st timestamp is calculated. We use that difference as an offset to update all the remaining timestamps. As such, while the timestamps are changed, the intervals between CAN frames are preserved; thus, we preserve the fidelity of the dataset.

Labeling the `.csv` files. We adopted two labeling techniques for our dataset. When labeling a given attack file, we selected the labeling technique best suited to the attack type (i.e., the labeling technique best suited to the characteristics of the injected attack CAN frames). When we simulate an attack, we begin with an attack-free traffic capture. We replay the attack-free traffic capture, inject attacks, and record the attack-laden traffic. Ultimately, we have an attack-free traffic capture and a corresponding attack-laden traffic capture. By comparing the attack-free capture to the attack-laden capture, we can identify the attack CAN frames. We use this labeling technique for DoS and fuzzing attacks, as the attack CAN frames do not resemble legitimate traffic. For spoofing attacks, however, the attack CAN frames closely

Table 7
Our datasets.

Name[a]	# of Lines[b]	Size	Format
`can-dataset`	332,022,322	13.3 GB	`.log` and `.csv`
`can-log (can-dataset)`	166,011,012	7.2 GB	`.log`
`can-csv (can-dataset)`	166,011,310	6.1 GB	`.csv`
`can-ml (.log files)`	315,295,222	13.8 GB	`.log`
`can-ml (.csv files)`	630,587,197	23.7 GB	`.csv`
`can-train-and-test`[c]	193,241,081	7.5 GB	`.csv`

[a] A dataset's "name" is also the name of its repository.
[b] The number of lines corresponds to the number of samples—or timesteps—in the dataset.
[c] `can-train-and-test` is the name of the repository which contains the final curated dataset (labeled and partitioned). The `can-train-and-test` repository is linked to the `can-dataset` and `can-ml` repositories, which contain raw `.log` files, additional attack-free traffic, and additional attack traffic. As such, we use the term `can-train-and-test` to refer to the `can-train-and-test` repository as well as the linked repositories.

resemble—or even match—legitimate traffic. When we replay and recapture CAN traffic, we inadvertently change the timestamps (both the actual dates/times and the intervals), so we cannot distinguish between attack-free and attack CAN frames using the aforementioned comparison technique. Instead, when injecting attack CAN frames, we use a different CAN interface (e.g., `can1`, `vcan1`). During the labeling process, we leverage the CAN interface to differentiate attack-free and attack traffic. Once the file is labeled, we standardize the CAN interface to eliminate noise. Ultimately, our labeled `.csv` files contain the following fields:

1. `timestamp`
2. `arbitration_id`
3. `data_field`
4. `attack` ("0" for attack-free, "1" for attack)

Curating the sub-datasets. We subdivided our dataset into four sub-datasets, each of which contains one "training" folder and four "testing" folders. The sub-datasets are intended to be similar in terms of size (e.g., number of samples) and attacks (e.g., attack-free/attack ratio). We planned out our sub-datasets by selecting different vehicle types, attack types, etc. for the various "training" and "testing" folders, then we constructed the sub-datasets accordingly. We discuss our sub-datasets in detail in Section 5.

4.5.1. *Attack-free*

Listing 11 contains an example of attack-free traffic—in particular, traffic collected in driving mode—in a labeled `.csv` file. This is the end result when all pre-processing steps have been completed. The traffic was collected from the 2011 Chevrolet Traverse. An extended listing is available in the appendix (see Section Appendix B.1).

Listing 11 Labeled attack-free traffic (driving mode).

```
1672531371.842441,1E1,0000100000,0
1672531371.843111,0C7,00D4A352,0
1672531371.843342,0F9,031A4002A23FDEFA,0
1672531371.843591,189,4FFF0FFF3000DEFA,0
1672531371.843815,199,4FFF0E70F18F00FF,0
1672531371.8439581,1EB,0141,0
1672531371.844667,1F1,AE0E00000800007A,0
1672531371.845175,0C1,236F7087236A5491,0
1672531371.8453221,1CB,10E600,0
1672531371.845534,0C5,234188A42341E475,0
1672531371.8457692,184,0001000001FF,0
1672531371.845994,1C7,0FFF700003FF3F,0
1672531371.846168,1CD,C7FF07FE7F,0
1672531371.846438,1E5,4600D8EED4FF2500,0
1672531371.8466392,1E9,0FFF000C00260000,0
1672531371.846887,0C9,801D551911000000,0
1672531371.847044,0F1,1C040040,0
1672531371.847256,191,075E075E0761115D,0
```

4.5.2. *Attack*

Listing 12 contains an example of attack traffic—specifically, fuzzing attack traffic—in a labeled `.csv` file. This is the end result when all pre-processing steps have been completed. The traffic was collected from the 2011 Chevrolet Traverse. An extended listing is available in the appendix (see Section Appendix B.2).

Listing 12 A labeled fuzzing attack. The red lines indicate attack CAN frames.

```
1672531209.2064872,348,00000000,0
1672531209.2064922,34A,00000000,0
1672531209.2072642,036,F237,1
1672531209.208342,0EE,750F037DB0973413,1
1672531209.208786,0F1,34050040,0
1672531209.2094111,63C,BB98131FFB6BB64D,1
1672531209.209938,19D,C0003FFD000000FF,0
1672531209.209954,1AF,000000,0
1672531209.210492,758,B4E4B84DB744242D,1
1672531209.2110262,1F5,0F0F002100000300,0
1672531209.211039,1EB,0000,0
1672531209.211636,1B9,619E732063822815,1
1672531209.2128332,107,A60ED87316918238,1
1672531209.213259,0C1,0000000000000000,0
1672531209.21398,3AA,9442CD7C6E7A727F,1
1672531209.2143612,0C5,000000000002A384,0
1672531209.214379,184,0002000001FE,0
1672531209.214384,0C9,0000000A00000000,0
```

5. Datasets

In this section, we introduce our datasets as shown in Table 7: `can-dataset`, `can-log`, `can-csv`, `can-ml`, and `can-train-and-test`. We will begin with the `can-dataset`, which contains raw CAN traffic data, and we will conclude with `can-train-and-test`, which is pre-processed, labeled, organized, and curated. Each of the five datasets will be presented in its own section.

5.1. `can-dataset`

`can-dataset` is the name of our raw CAN dataset, available at https://bitbucket.org/brooke-lampe/can-dataset/src/master/ or https://data.dtu.dk/articles/dataset/can-dataset/24805248. This dataset contains 596 files (298 `.log` files and 298 `.csv` files), which consume 13.3 gigabytes (GB) of space.

The total number of lines in this dataset is ***332,022,322.*** Note that this dataset provides raw CAN data in two different formats—`.log` and `.csv`. As such, the number of unique samples is approximately half the total number of lines (Lampe, 2023b,g).

5.2. `can-log`

`can-log` is a subset of the `can-dataset`, available at https://bitbucket.org/brooke-lampe/can-log/src/master/ or https://data.dtu.dk/articles/dataset/can-log/24805506. It contains only the `.log` files and is available for researchers who do not want to download the full `can-dataset`, which includes both the `.log` and `.csv` files.

`can-log` contains 298 `.log` files, which consume 7.2 GB of space. The total number of lines (i.e., samples) in this dataset is ***166,011,012.*** Each sample corresponds to a particular CAN frame at a particular time (Lampe, 2023c,h).

There are 91,827,504 attack-free samples. Breaking that number down, we have...

1.	Chevrolet Impala	**14,693,040** attack-free samples
2.	Chevrolet Traverse	**53,086,095** attack-free samples
3.	Chevrolet Silverado	**12,867,782** attack-free samples
4.	Subaru Forester	**11,180,587** attack-free samples

5.3. `can-csv`

`can-csv` is a subset of the `can-dataset`, available at https://bitbucket.org/brooke-lampe/can-csv/src/master/ or https://data.dtu.dk/articles/dataset/can-csv/24805509. It contains only the `.csv` files. Similar to `can-log`, `can-csv` is available for researchers who do not want to download the full `can-dataset`, which includes both the `.log` and `.csv` files.

`can-csv` contains 298 `.csv` files, which consume 6.1 GB of space. The total number of lines (i.e., samples) in this dataset is ***166,011,310.*** As with `can-log`, each sample corresponds to a particular CAN frame at a particular time (Lampe, 2023a,f).

5.4. `can-ml`

We gave the name `can-ml` to our pre-processed, labeled dataset, as it is intended to support machine learning (i.e., ML)—specifically for intrusion detection. This dataset is available at https://bitbucket.org/brooke-lampe/can-ml/src/master/ or https://data.dtu.dk/articles/dataset/can-ml/24805530.

`can-ml` has been subdivided into four directories, as follows:

1. *Pre*-attack unlabeled
2. *Pre*-attack labeled
3. *Post*-attack unlabeled
4. *Post*-attack labeled

Essentially, for each traffic capture, we provide both labeled and unlabeled (raw) variants as well as pre-attack (attack-free) and attack variants. The bulk of the files are `.csv` files, since the comma-separated values (CSV) format is well suited to data labeling—and data loading. Our `.csv` files can be easily loaded with, e.g., the `pandas` (pandas Contributors, 2023) Python library. In the unlabeled pre-attack and post-attack directories, we also include the corresponding raw `.log` files (Lampe, 2023d,i).

Discounting the helper files, our dataset contains 1,248 files—`.csv` and `.log`—and occupies 37.5 GB of space. There are 312 files per vehicle (Chevrolet Impala, Chevrolet Traverse, etc.). Breaking it down, we have...

1. **`.log` files.** There are 416 `.log` files, totaling 13.8 GB. 104 `.log` files are associated with each vehicle. Summing up all the samples in all the `.log` files gives us a total of ***315,295,222*** samples.
2. **`.csv` files.** There are 832 `.csv` files, totaling 23.7 GB. 208 `.csv` files are associated with each vehicle. Summing up all the samples in all the `.csv` files gives us a total of ***630,587,197*** samples.

The `can-ml` dataset contains nine distinct attacks (see Section 4.3). Table 8 records the number of attack-free instances and the number of attack instances (i.e., CAN frames) for each attack. Some types of attacks, such as DoS, typically contain a very high number of attack instances—exactly what we would expect for that type of attack. Others, such as speed spoofing, can be more subtle, producing real-world consequences with a few thousand CAN frames.

5.5. `can-train-and-test`

`can-train-and-test` is our curated CAN intrusion detection dataset for machine learning IDSs. This dataset is (1) pre-processed, (2) labeled, and (3) organized. `can-train-and-test` is available at https://bitbucket.org/brooke-lampe/can-train-and-test/src/master/ or https://data.dtu.dk/articles/dataset/can-train-and-test/24805533.

`can-train-and-test` contains 236 `.csv` files and comprises a total of ***193,241,081*** lines, which corresponds to 7.5 GB of space (Lampe, 2023e,j). It is subdivided into four train/test sub-datasets, as follows:

1. **`set_01`**
   - 52 `.csv` files
   - 1.7 GB of space
   - 10,653,152 training samples
   - 44,659,609 total samples
2. **`set_02`**
   - 56 `.csv` files
   - 2.2 GB of space
   - 17,340,826 training samples
   - 55,946,808 total samples
3. **`set_03`**
   - 62 `.csv` files
   - 1.7 GB of space
   - 12,025,781 training samples
   - 43,799,824 total samples
4. **`set_03`**
   - 66 `.csv` files
   - 1.9 GB of space
   - 9,492,819 training samples
   - 48,834,840 total samples

Within each train/test sub-dataset—`set_01`, `set_02`, `set_03`, and `set_04`—we provide one training subset and four testing subsets, as follows:

1. **train_01:** Train the model
2. **test_01_known_vehicle_known_attack:** Test the model against a known vehicle (seen in training) and known attacks (seen in training)
3. **test_02_unknown_vehicle_known_attack:** Test the model against an unknown vehicle (not seen in training) and known attacks (seen in training)
4. **test_03_known_vehicle_unknown_attack:** Test the model against a known vehicle (seen in training) and unknown attacks (not seen in training)
5. **test_04_unknown_vehicle_unknown_attack:** Test the model against an unknown vehicle (not seen in training) and unknown attacks (not seen in training)

5.5.1. *Class imbalance*

Now, we will discuss class balance—or rather, class *im*balance—in the `can-train-and-test` dataset, which is the dataset we will use for our benchmark in Section 7. Our problem is a *binary classification* problem—we assign samples to one of two categories. A sample can either be an *attack-free* sample or an *attack* sample (i.e., a positive sample or a negative sample). In a *balanced classification* problem, the class distribution is roughly equal; that is, there are about as many positive

Table 8
Attack-free and attack instances.

Vehicle	Attack Type	# of Attack-Free Instances	# of Attack Instances
2011 Chevrolet Impala	Denial of Service (DoS)	1,385,357	90,462
—	Combined Spoofing	6,737,943	27,042
—	Fuzzing	1,221,636	53,140
—	Gear Spoofing	3,448,455	9,483
—	Interval	1,955,381	7,126
—	RPM Spoofing	4,633,645	10,776
—	Speed Spoofing	4,736,665	14,147
—	Standstill	4,467,371	5,259
—	Systematic	1,734,400	31,085
2011 Chevrolet Traverse	Denial of Service (DoS)	2,231,741	79,900
—	Combined Spoofing	8,940,577	52,580
—	Fuzzing	3,392,420	125,942
—	Gear Spoofing	7,829,122	1,216
—	Interval	4,020,305	16,437
—	RPM Spoofing	9,830,326	931
—	Speed Spoofing	12,127,282	3,347
—	Standstill	3,306,519	609
—	Systematic	1,795,032	40,365
2016 Chevrolet Silverado	Denial of Service (DoS)	2,226,833	261,604
—	Combined Spoofing	7,214,913	11,713
—	Fuzzing	2,889,336	55,095
—	Gear Spoofing	4,268,509	3,772
—	Interval	3,313,734	9,977
—	RPM Spoofing	3,925,254	1,979
—	Speed Spoofing	5,247,359	5,655
—	Standstill	2,946,404	2,454
—	Systematic	3,249,507	20,915
2017 Subaru Forester	Denial of Service (DoS)	2,246,099	51,798
—	Combined Spoofing	5,722,160	84,665
—	Fuzzing	2,413,372	37,391
—	Gear Spoofing	2,348,436	4,677
—	Interval	2,517,118	106,314
—	RPM Spoofing	2,974,813	16,829
—	Speed Spoofing	3,654,235	18,262
—	Standstill	2,350,664	10,808
—	Systematic	3,374,832	23,814

samples as negative samples. Our problem is an *imbalanced classification* problem (Brownlee, 2020; Kubat and Matwin, 1997).

In automotive networks, upwards of 1 million messages cross the CAN bus every ten minutes. Automobiles travel for miles and miles, for hours upon hours, without falling victim to an attack. Therefore, automotive intrusion detection is a problem of rare event prediction; the rare event in question is automotive attack. Our problem is inherently imbalanced; therefore, our dataset is similarly imbalanced. A practicable IDS would have to be able to cope with this imbalance—whether it is pitted against our dataset or placed in a real automobile.

For each sub-dataset and training/testing subset, the ratio of attack-free traffic to attack traffic (`attack-free:attack`) is presented below:

1. **Sub-dataset #1**
   - Training subset - **213:1**
   - Testing subset #1 - **89:1**
   - Testing subset #2 - **38:1**
   - Testing subset #3 - **413:1**
   - Testing subset #4 - **927:1**
2. **Sub-dataset #2**
   - Training subset - **76:1**
   - Testing subset #1 - **927:1**
   - Testing subset #2 - **420:1**
   - Testing subset #3 - **421:1**
   - Testing subset #4 - **264:1**
3. **Sub-dataset #3**
   - Training subset - **69:1**
   - Testing subset #1 - **51:1**
   - Testing subset #2 - **42:1**
   - Testing subset #3 - **519:1**
   - Testing subset #4 - **44:1**
4. **Sub-dataset #4**
   - Training subset - **363:1**
   - Testing subset #1 - **44:1**
   - Testing subset #2 - **303:1**
   - Testing subset #3 - **42:1**
   - Testing subset #4 - **36:1**

Reviewing the above ratios, we can see that class imbalances range from a ratio of **36:1** at the lowest to ratio of **927:1** at the highest. When we present our benchmark (Section 7), we will revisit class imbalance and the complications it introduces.

The `can-train-and-test` dataset is a curated subset of the `can-ml` dataset, which, in turn, is the pre-processed counterpart of the raw `can-dataset`. As such, class imbalance in the can-train-and-test dataset is representative of class imbalance in the `can-dataset` and `can-ml` datasets.

6. Practitioner guidance

As emphasized in Section 5, we provide multiple datasets—in multiple repositories—for different use cases. In this section, we highlight the features and use cases of our datasets to help practitioners decide which dataset will best suit their purposes. We also provide several usage examples that should help practitioners to get started working with our datasets (see Section 6.1).

Table 9 illustrates our datasets and their expected use cases.

The links to all of our repositories are as follows:

Table 9
Our datasets: *Practitioner Guidance.*

Name[a]	Use Case(s)	Notes
can-dataset	This dataset is a superset of the can-log and can-csv datasets. Practitioners who plan to use both the can-log and can-csv datasets might find it easier to download this dataset, as all of the .log and .csv are stored side by side in the appropriate directories.	can-dataset is a repository that contains all of the contents of both the can-log and can-csv repositories.
can-log (can-dataset)	This data contains raw CAN traffic in .log files. Practitioners can use it to replay CAN traffic (see Section 6.1). During replay, practitioners can also inject CAN frames to develop new attacks.	can-log is the .log-file only version of the can-dataset repository.
can-csv (can-dataset)	This dataset contains raw CAN traffic that has been converted from .log files to .csv files. It is intended to support unsupervised learning, as it is unlabeled, but it is in a format that can be quickly and efficiently loaded into a Python program (e.g., using the pandas pandas Contributors (2023) library).	can-csv is the .csv-file only version of the can-dataset repository.
can-ml (.log and .csv files)	This dataset contains labeled .csv files for attack-free traffic as well as nine types of attack traffic. It is intended to support supervised machine learning. Artifacts used to generate the labeled .csv files (e.g., the original .log files) are also included.	If an attack was conducted offline rather than in-vehicle, then the original .log files will be available under the directory "pre-attack-unlabeled."
can-train-and-test	This dataset contains four curated sub-datasets. The sub-datasets have been subdivided into one training subset and four testing subsets. The testing subsets correspond to different attack scenarios (e.g., known vehicle, unknown attack). Practitioners interested in machine learning can use this pre-split dataset in order to avoid partitioning the can-ml dataset themselves. Practitioners interested in seeing how well an IDS generalizes to unseen attacks or unseen vehicles might also wish to use this dataset.	This dataset is a subset of the can-ml dataset; it does not contain all the CAN traffic data we collected.

[a] A dataset's "name" is also the name of its repository.

- **can-dataset**: https://bitbucket.org/brooke-lampe/can-dataset/src/master/, https://data.dtu.dk/articles/dataset/can-dataset/24805248, or https://doi.org/10.11583/DTU.24805248.
- **can-log**: https://bitbucket.org/brooke-lampe/can-log/src/master/, https://data.dtu.dk/articles/dataset/can-log/24805506, or https://doi.org/10.11583/DTU.24805506.
- **can-csv**: https://bitbucket.org/brooke-lampe/can-csv/src/master/, https://data.dtu.dk/articles/dataset/can-csv/24805509, or https://doi.org/10.11583/DTU.24805509.
- **can-ml**: https://bitbucket.org/brooke-lampe/can-ml/src/master/, https://data.dtu.dk/articles/dataset/can-ml/24805530, or https://doi.org/10.11583/DTU.24805530.
- **can-train-and-test**: https://bitbucket.org/brooke-lampe/can-train-and-test/src/master/, https://data.dtu.dk/articles/dataset/can-train-and-test/24805533, or https://doi.org/10.11583/DTU.24805533.

6.1. Usage

We provide a number of usage examples for the raw CAN traffic logs—i.e., the .log files. Our usage examples require Linux's SocketCAN subsystem (Community, 2023) and can-utils utilities (can-utils Contributors, 2023). We include the commands to (1) set up a virtual CAN interface and (2) replay the raw CAN traffic logs (using canplayer).

```
# Set up virtual CAN interface
sudo modprobe vcan
sudo ip link add dev vcan0 type vcan
sudo ip link set vcan0 up

# Show the details of the vcan0 link
ip -details -statistics link show vcan0

# Replay a candump log file
canplayer -I your-log-file.log vcan0=can0

# To replay an attack-free log from the
  ↪ can-dataset
canplayer -I can-dataset/2017-subaru-
  ↪ forester/attack-free/attack-free-1.log
  ↪ vcan0=can0

# To replay a DoS log from the can-dataset
canplayer -I can-dataset/2011-chevrolet-
  ↪ impala/DoS-attacks/DoS-1.log
  ↪ vcan0=can0

# Examine packets with the specified arbitration
  ↪ identifier
candump vcan0 | grep " 3E9 "

# Use candump to create a log file that can be
  ↪ replayed using canplayer
candump -l any

# Use candump to create a log file that can be
  ↪ replayed using canplayer,
# specifying the log file name
candump -L any > your-log-file.log

# Use candump to create a log file that can be
  ↪ replayed using canplayer,
# specifying the virtual CAN interface
candump -L vcan0 > DoS-1.log

# Send spoofed packets at set intervals
while true; do cansend vcan0
  ↪ 3E9#1B4C05111B511C69; sleep 0.01; done

# Monitor the bus; send spoofed packets when
  ↪ real packets are detected
candump can0 | grep " 3E9 " | while read line;
  ↪ do cansend can0 3E9#1B4C05111B511C69; done
```

7. Benchmark

7.1. Setup

We implemented our benchmark in Python, leveraging the pandas (pandas Contributors, 2023) Python library to load our dataset and the scikit-learn (Pedregosa et al., 2011) Python library to implement our machine learning intrusion detection systems. We selected several machine learning algorithms—both supervised and unsupervised—as well as two deep learning algorithms from scikit-learn's offerings.

Table 10
Sub-dataset #1, Testing subset #1 (known vehicle, known attack).

Model	Accuracy	Precision	Recall (TPR)	F1-score	Training Time (ns)	Testing Time (ns)
Gaussian Naive Bayes	0.8820	0.9814	0.8820	0.9273	2071507728	617191930
K-Nearest Neighbors	0.9855	0.3376	0.3141	0.3254	106205148337753	2698926122173
Linear Regression	0.3540	0.9680	0.3540	0.5042	3544294407	141826800
Logistic Regression	0.9890	0.9871	0.9890	0.9878	112252658417	380374780
Linear Support Vector Machine	0.9841	0.3018	0.3275	0.3141	35314797566	344383409
One-Class Support Vector Machine[a]	-	-	-	-	-	-
Support Vector Machine	0.9720	0.1505	0.3275	0.2062	116557173711567	20169585048115
Decision Tree	0.9742	0.0724	0.1123	0.0880	8271963209	401618047
Extra Trees	0.9915	0.8199	0.3046	0.4442	209531429363	20667658310
Gradient Boosting	0.6254	0.9892	0.6254	0.7585	2451403662347	8489738261
Isolation Forest	0.9846	0.9779	0.9846	0.9812	197244480367	88099567128
Random Forest	0.9879	0.0088	0.0008	0.0015	763511969876	38249637250
K-Means Clustering	0.8564	0.9763	0.8564	0.9124	97416359356	638434302
Mini-Batch K-Means Clustering	0.4681	0.9669	0.4681	0.6304	7678285764	3796294688
BIRCH	0.9889	0.9779	0.9889	0.9834	214197454658	605665184
Local Outlier Factor	0.9254	0.9728	0.9254	0.9465	53788202512612	2439824879120
Multi-Layer Perceptron	0.9788	0.9837	0.9788	0.9811	3088090156514	11708139937
Restricted Boltzmann Machine	0.0111	0.0001	0.0111	0.0002	916661700320	9736619899

[a] The one-class support vector machine exceeded our 6-day (518400000000000 ns) time limit.

For supervised traditional machine learning, we experimented with five families of models—Gaussian naive Bayes (1 model), k-nearest neighbors (1 model), regression (2 models), support vector machine (3 models), and tree (5 models). In total, there are twelve supervised machine learning models. For unsupervised machine learning, we explored two families of models—clustering (3 models) and local outlier factor (1 model). We evaluated one supervised machine learning model—a multi-layer perceptron—and one unsupervised deep learning model—a restricted Boltzmann machine.

- **SUPERVISED TRADITIONAL MACHINE LEARNING**
  - Gaussian Naive Bayes (NB)
  - K-Nearest Neighbors (KNN)
  - Regression
    - *Linear Regression*
    - *Logistic Regression*
  - Support Vector Machine (SVM)
    - *SVM*
    - *Linear SVM*
    - *One-Class SVM*
  - Tree
    - *Decision Tree (DT)*
    - *Extra Trees (ET)*
    - *Gradient Boosting*
    - *Isolation Forest (IF)*
    - *Random Forest (RF)*
- **UNSUPERVISED TRADITIONAL MACHINE LEARNING**
  - Clustering
    - *K-Means Clustering*
    - *Mini-Batch K-Means Clustering*
    - *Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)*
  - Local Outlier Factor (LOF)
- **SUPERVISED DEEP LEARNING**
  - Multi-Layer Perceptron (MLP)
- **UNSUPERVISED DEEP LEARNING**
  - Restricted Boltzmann Machine (RBM)

We leveraged a compute cluster to conduct our experiments. Each experiment was allocated one CPU and 16 GB of RAM. The time limit was set to 6 days (518400000000000 ns).

7.2. Discussion of results

In this section, we discuss the results of our evaluation. We selected sub-dataset #1 to examine in detail; additional results have been made available in the appendix (see Appendix C) as well as the `can-benchmark` repository.

7.2.1. Testing subsets

Tables 10, 11, 12, 13 showcase our experimental results for sub-dataset #1, testing subsets #1, #2, #3, and #4, respectively. For each model, the experiment with the optimal parameter set—in terms of maximum F1-score—is shown.

Recall that `test_01` is designed to evaluate a given model against a *known* vehicle and *known* attacks. Next, `test_02` evaluates the model against an *unknown* vehicle and *known* attacks. Inversely, `test_03` evaluates the model against a *known* vehicle and *unknown* attacks. Lastly, `test_04` evaluates the model against an *unknown* vehicle and *unknown* attacks.

Looking at testing subset #1 (Table 10), we can see that the logistic regression model achieved the highest F1-score: 0.9878. The logistic regression model performed very well across the board; its accuracy, precision, and recall metrics were 0.9890, 0.9871, and 0.9890, respectively. Training time was approximately 112.3 seconds; testing time was approximately 0.38 seconds.

For testing subset #2 (Table 11), the BIRCH model earned the highest F1-score: 0.9620—somewhat lower than the highest F1-score for the previous testing subset. The BIRCH model also attained an accuracy of 0.9745, a precision of 0.9497, and a recall of 0.9745. The BIRCH model spent approximately 214.2 seconds on training and 0.66 seconds on testing.

Next, we examine testing subset #3 (Table 12). We can see that the gradient boosting model attained the highest F1-score: 0.9966. The gradient boosting model attained an accuracy of 0.9977, a precision of 0.9977, and a recall of 0.9977. Training time was approximately 2451.4 seconds (≈41 minutes); testing time was approximately 11.8 seconds.

Table 11
Sub-dataset #1, Testing subset #2 (unknown vehicle, known attack).

Model	Accuracy	Precision	Recall (TPR)	F1-score	Training Time (ns)	Testing Time (ns)
Gaussian Naive Bayes	0.8652	0.9479	0.8652	0.9044	2071507728	674632590
K-Nearest Neighbors	0.9714	0.0009	0.0001	0.0002	106205148337753	6628601627607
Linear Regression	0.3112	0.9668	0.3112	0.4499	3544294407	127655570
Logistic Regression	0.9565	0.9514	0.9565	0.9539	112252658417	199207470
Linear Support Vector Machine	0.9593	0.0538	0.0361	0.0432	35314797566	354491014
One-Class Support Vector Machine[a]	-	-	-	-	-	-
Support Vector Machine	0.9720	0.1505	0.3275	0.2062	116557173711567	20169585048115
Decision Tree	0.9738	0.0000	0.0000	0.0000	8271963209	471393563
Extra Trees	0.9740	0.0000	0.0000	0.0000	209531429363	24155191698
Gradient Boosting	0.6626	0.9763	0.6626	0.7739	2451403662347	8834463042
Isolation Forest	0.9550	0.9492	0.9550	0.9521	197244480367	100337823607
Random Forest	0.9720	0.0000	0.0000	0.0000	763511969876	42700298556
K-Means Clustering	0.8654	0.9467	0.8654	0.9042	97416359356	709549141
Mini-Batch K-Means Clustering	0.4969	0.9271	0.4969	0.6470	7678285764	4126732218
BIRCH	0.9745	0.9497	0.9745	0.9620	214197454658	662717530
Local Outlier Factor	0.1714	0.9557	0.1714	0.2640	53788202512612	7698684672802
Multi-Layer Perceptron	0.9666	0.9529	0.9666	0.9594	3088090156514	13276672102
Restricted Boltzmann Machine	0.0255	0.0006	0.0255	0.0013	916661700320	10970949396

[a] The one-class support vector machine exceeded our 6-day (518400000000000 ns) time limit.

Table 12
Sub-dataset #1, Testing subset #3 (known vehicle, unknown attack).

Model	Accuracy	Precision	Recall (TPR)	F1-score	Training Time (ns)	Testing Time (ns)
Gaussian Naive Bayes	0.8839	0.9957	0.8839	0.9361	2071507728	1069338929
K-Nearest Neighbors	0.9976	0.4762	0.0005	0.0010	106205148337753	3326301454816
Linear Regression	0.3467	0.9939	0.3467	0.5136	3544294407	159399960
Logistic Regression	0.9976	0.9954	0.9976	0.9964	112252658417	696710769
Linear Support Vector Machine	0.9975	0.0131	0.0009	0.0017	35314797566	673546732
One-Class Support Vector Machine[a]	-	-	-	-	-	-
Support Vector Machine	0.9976	0.0260	0.0009	0.0018	116557173711567	30536179241509
Decision Tree	0.9821	0.0022	0.0148	0.0039	8271963209	602445631
Extra Trees	0.9976	1.0000	0.0001	0.0003	209531429363	31514357019
Gradient Boosting	0.9977	0.9977	0.9977	0.9966	2451403662347	11825961968
Isolation Forest	0.9969	0.9957	0.9969	0.9963	197244480367	134518109020
Random Forest	0.9976	1.0000	0.0004	0.0009	763511969876	56868215083
K-Means Clustering	0.8658	0.9970	0.8658	0.9258	97416359356	963140938
Mini-Batch K-Means Clustering	0.4741	0.9969	0.4741	0.6408	7678285764	5800975603
BIRCH	0.9976	0.9953	0.9976	0.9965	214197454658	1004912418
Local Outlier Factor	0.9386	0.9965	0.9386	0.9665	53788202512612	2504114371275
Multi-Layer Perceptron	0.9974	0.9953	0.9974	0.9964	3088090156514	17550210412
Restricted Boltzmann Machine	0.0024	0.0000	0.0024	0.0000	916661700320	14603996262

[a] The one-class support vector machine exceeded our 6-day (518400000000000 ns) time limit.

Lastly, looking at testing subset #4 (Table 13), we can see that, once again, the BIRCH model earned the highest F1-score: 0.9984. For accuracy, precision, and recall, the BIRCH model attained values of 0.9990, 0.9979, and 0.9990, respectively. As before, the training time was approximately 214.2 seconds, though the testing time was marginally longer—approximately 1.2 seconds.

If we zoom out to the full dataset—which includes all experiments with all parameters—we can see a trend in terms of the highest-performing models. Generally, the following models achieve the top F1-scores:

1. Multi-Layer Perceptron
2. Gradient Boosting
3. Isolation Forest
4. BIRCH
5. Logistic Regression

On the flip side, the following models earned F1-scores of *zero* for one or more experiments:

- Restricted Boltzmann Machine
- Decision Tree
- Extra Trees
- K-Nearest Neighbors
- Linear Support Vector Machine
- Support Vector Machine
- Random Forest

We do not have an F1-score for the one-class support vector machine—as it ran out of time—but, given the performance of the linear support vector machine and the traditional support vector machine, we do not believe that it would have performed well.

When evaluating the performance of a machine learning model; training time is also a very important metric. From a training time per-

Table 13
Sub-dataset #1, Testing subset #4 (unknown vehicle, unknown attack).

Model	Accuracy	Precision	Recall (TPR)	F1-score	Training Time (ns)	Testing Time (ns)
Gaussian Naive Bayes	0.8906	0.9979	0.8906	0.9411	2071507728	1671649398
K-Nearest Neighbors	0.9982	0.0190	0.0148	0.0166	106205148337753	11972712642811
Linear Regression	0.3341	0.9758	0.3341	0.4757	3544294407	137564940
Logistic Regression	0.9976	0.9979	0.9976	0.9977	112252658417	984385909
Linear Support Vector Machine	0.9942	0.0002	0.0009	0.0003	35314797566	661366602
One-Class Support Vector Machine[a]	-	-	-	-	-	-
Support Vector Machine	0.9981	0.0007	0.0006	0.0007	116557173711567	46633303631277
Decision Tree	0.9827	0.0018	0.0276	0.0033	8271963209	997735598
Extra Trees	0.9988	0.0000	0.0000	0.0000	209531429363	49439825750
Gradient Boosting	0.9984	0.9980	0.9984	0.9982	2451403662347	18095133741
Isolation Forest	0.9881	0.9980	0.9881	0.9930	197244480367	202233230770
Random Forest	0.9988	0.0011	0.0001	0.0003	763511969876	88901090110
K-Means Clustering	0.8872	0.9985	0.8872	0.9392	97416359356	1463804724
Mini-Batch K-Means Clustering	0.4748	0.9984	0.4748	0.6429	7678285764	9274790788
BIRCH	0.9990	0.9979	0.9990	0.9984	214197454658	1171693676
Local Outlier Factor	0.1661	0.8882	0.1661	0.2726	53788202512612	7995192332965
Multi-Layer Perceptron	0.9973	0.9979	0.9973	0.9976	3088090156514	27715472855
Restricted Boltzmann Machine	0.0010	0.0000	0.0010	0.0000	916661700320	22041127093

[a] The one-class support vector machine exceeded our 6-day (518400000000000 ns) time limit.

spective, the Gaussian naive Bayes model is the clear victor. Looking at sub-dataset #4, testing subset #2, not only did the Gaussian naive Bayes model achieve the shortest training time—1711150808 nanoseconds (≈1.7 seconds)—but it also earned an impressive 0.9955 F1-score. Moreover, the model achieved an accuracy of 0.9961, a precision of 0.9951, and a recall of 0.9961. The worst performer, time-wise, would be the one-class support vector machine, as it failed to terminate within the 6-day time limit.

As mentioned above, the testing subsets were developed to evaluate different facets of a machine learning model. Some subsets evaluate the given model against *knowns* (i.e., known vehicles, known attacks), while others evaluate the given model against *unknowns* (i.e., unknown vehicles, unknown attacks). The average F1-scores for our testing subsets are given below:

1. Testing subset #1	**F1-score:** 0.49833623
2. Testing subset #2	**F1-score:** 0.42135362
3. Testing subset #3	**F1-score:** 0.45809565
4. Testing subset #4	**F1-score:** 0.44716087
5. All testing subsets	**F1-score:** 0.45623659

The average F1-score varied across testing subsets #1 through #4. The average F1-score for testing subset #1 was ≈0.4983. For testing subset #2, the average F1-score dropped to ≈0.4214. For testing subsets #3 and #4, the average F1-scores were ≈0.4581 and ≈0.4472, respectively. The average F1-score across all testing subsets was ≈0.4562.

Unsurprisingly, the average F1-score was highest for testing subset #1, which contains only known vehicles and known attacks. We expected testing subset #4, which contains both unknown vehicles and unknown attacks, to attain the lowest average F1-score. However, testing subset #2 (unknown vehicles, known attacks) saw the lowest average F1-score. It would seem that unknown vehicles might be more disruptive to machine learning IDSs than unknown attacks. If that is indeed the case, then it is all the more important to train a machine learning IDS against as many different vehicles (type, manufacturer, model) as possible.

7.2.2. Sub-datasets

With our four testing subsets, our objective was to evaluate different aspects of a machine learning model. Thus, the testing subsets were carefully designed to be *different*. With our four sub-datasets, our objective was *similarity*—we sought to construct sub-datasets that were consistent in terms of size, complexity, difficulty, attack types, etc. Sub-datasets provide supplementary training and testing data; for example, if sub-dataset #1 is too small, it can be augmented with data from sub-dataset #2. In addition, sub-datasets could be used to ensure that a machine learning model does not suffer from overfitting; if the model performs well on sub-dataset #1 but performs terribly on sub-datasets #2, #3, and #4, then perhaps the model might have overfit to sub-dataset #1.

Therefore, when we compare sub-datasets, we hope to see similar metrics for the same model when evaluated against the same testing subset. Let us look at the BIRCH model when evaluated against testing subset #1 for both sub-dataset #1 and sub-dataset #4. For sub-dataset #1, accuracy, precision, recall, and F1-score were 0.9889, 0.9779, 0.9889, and 0.9834, respectively. Meanwhile, for sub-dataset #4, accuracy, precision, recall, and F1-score were 0.9784, 0.9572, 0.9784, and 0.9677, respectively. Across the board, the values are somewhat lower for sub-dataset #4 than for sub-dataset #1. That said, the BIRCH model performed exceptionally well on both sub-datasets. The training times were 214197454658 ns (≈214.2 seconds) and 161254066683 ns (≈161.3 seconds) for sub-dataset #1 and sub-dataset #4, respectively. Overall, the two sub-datasets are sufficiently similar.

On the opposite end of the spectrum, the k-nearest neighbors model performed poorly when pitted against sub-dataset #1, testing subset #1, achieving an F1-score of only 0.3254. Similarly, the k-nearest neighbors model underperformed against sub-dataset #4, testing subset #1, earning an F1-score of only 0.3279. Time-wise, the model spent 106205148337753 ns (≈29.5 hours) training against sub-dataset #1 and 68650714770712 ns (≈19.1 hours) training against sub-dataset #4.

Table 14 highlights the performance of our 18 models against sub-dataset #4, testing subset #1. For comparison, refer to Table 10, which showcases the performance of our 18 models against sub-dataset #1, testing subset #1.

Calculating the average F1-score for each sub-dataset yields the following values:

1. Sub-dataset #1	**F1-score:** 0.45623659
2. Sub-dataset #2	**F1-score:** 0.49727649
3. Sub-dataset #3	**F1-score:** 0.52929375

Table 14
Sub-dataset #4, Testing subset #1 (known vehicle, known attack).

Model	Accuracy	Precision	Recall (TPR)	F1-score	Training Time (ns)	Testing Time (ns)
Gaussian Naive Bayes	0.9755	0.9596	0.9755	0.9668	1711150808	716855819
K-Nearest Neighbors	0.9825	0.9638	0.1975	0.3279	68650714770712	1751927202489
Linear Regression	0.3243	0.9985	0.3243	0.4886	4962548165	255239910
Logistic Regression	0.9784	0.9788	0.9784	0.9677	35900328378	199674490
Linear Support Vector Machine	0.9784	0.0000	0.0000	0.0000	21097322961	415427420
One-Class Support Vector Machine[a]	-	-	-	-	-	-
Support Vector Machine	0.9810	0.9913	0.1254	0.2226	89048060616553	7497330386451
Decision Tree	0.9880	0.9705	0.4586	0.6229	23308577039	687957287
Extra Trees	0.9837	0.9972	0.2467	0.3956	486225104765	36751033345
Gradient Boosting	0.9800	0.9771	0.9800	0.9722	2246450290838	7599878712
Isolation Forest	0.9774	0.9578	0.9774	0.9672	171923906654	103616521318
Random Forest	0.9848	0.9977	0.2961	0.4566	1658659682691	59969033775
K-Means Clustering	0.7698	0.9530	0.7698	0.8511	49276596864	757799028
Mini-Batch K-Means Clustering	0.7597	0.9526	0.7597	0.8448	7006960680	4772947871
BIRCH	0.9784	0.9572	0.9784	0.9677	161254066683	678632660
Local Outlier Factor	0.9552	0.9981	0.9552	0.9760	100295280221578	4241296731211
Multi-Layer Perceptron	0.9799	0.9787	0.9799	0.9715	2655845001046	13873431844
Restricted Boltzmann Machine	0.1171	0.9575	0.1171	0.1781	761974625735	12918669375

[a] The one-class support vector machine exceeded our 6-day (518400000000000 ns) time limit.

4. Sub-dataset #4 **F1-score:** 0.53296775
5. All sub-datasets **F1-score:** 0.50389963

Sub-dataset #1 appears to be somewhat more difficult than the others—the average F1-score of sub-dataset #1 is ≈0.4562, somewhat lower than the overall average F1-score of ≈0.5039. Judging by F1-score, sub-datasets #3 and #4 are remarkably similar, while sub-dataset #2 appears to be marginally more difficult than the overall dataset.

7.2.3. Different vehicles

To determine which machine learning model performed best across different vehicles, we need to know which vehicles constituted the known and unknown vehicles in each sub-dataset:

1. **Sub-dataset #1**
   - Known vehicle: *Chevrolet Impala*
   - Unknown vehicle: *Chevrolet Silverado*
2. **Sub-dataset #2**
   - Known vehicle: *Chevrolet Traverse*
   - Unknown vehicle: *Subaru Forester*
3. **Sub-dataset #3**
   - Known vehicle: *Chevrolet Silverado*
   - Unknown vehicle: *Subaru Forester*
4. **Sub-dataset #4**
   - Known vehicle: *Subaru Forester*
   - Unknown vehicle: *Chevrolet Traverse*

For each sub-dataset, we are interested in how well the machine learning model performs in the known vehicle scenarios, namely, testing subset #1 (known vehicle, known attack) and testing subset #3 (known vehicle, unknown attack). These testing subsets will demonstrate a machine learning model's capabilities across CAN traffic captures from different vehicles, whereas testing subsets #2 (unknown vehicle, known attack) and #4 (unknown vehicle, unknown attack) would instead demonstrate the ability of an IDS to generalize to different vehicles (vehicles that did not appear in training).

When determining the top machine learning models for each sub-dataset, testing subset pair, we include only models that *consistently* achieved a high F1-score; that is, models that achieved top scores two or more times. For testing subset #1 (known vehicle, known attack), the top two machine learning models—in terms of F1-score—are as follows:

1. **Sub-dataset #1:** MLP, BIRCH
2. **Sub-dataset #2:** Gradient Boosting, MLP (tie)
3. **Sub-dataset #3:** Gradient Boosting, MLP
4. **Sub-dataset #4:** MLP, Gradient Boosting

Then, for testing subset #3 (known vehicle, unknown attack), the top two machine learning models—in terms of F1-score—are as follows:

1. **Sub-dataset #1:** Gradient Boosting, Isolation Forest (tie)
2. **Sub-dataset #2:** Isolation Forest, BIRCH (tie)
3. **Sub-dataset #3:** Gradient Boosting, MLP
4. **Sub-dataset #4:** Gradient Boosting, MLP

We can see that the gradient boosting model and the multi-layer perceptron model perform very well regardless of vehicle model and, importantly, regardless of vehicle manufacturer. In sub-datasets #1, #2, and #3, the manufacturer is the same (Chevrolet), and both the gradient boosting model and the multi-layer perceptron model perform superbly—even when confronted with unknown attacks. Looking at sub-dataset #4, in which the manufacturer is different (Subaru), we can see that the gradient boosting model and the multi-layer perceptron model perform very well against both known and unknown attacks. Clearly, the gradient boosting and multi-layer perceptron models perform very well across different vehicles. However, in many cases, the difference in F1-score between 1st place and 2nd, 3rd, or even 4th place is quite small. For example, the BIRCH model tended to be in 3rd or 4th place—and the gap was often very small. Indications are that the gradient boosting and multi-layer perceptron models outperform the others across different vehicles, but with additional optimization and tuning—of both the models and the features—the rankings might change.

7.2.4. Traditional machine learning vs. deep learning, supervised learning vs. unsupervised learning

Our evaluation pitted (1) traditional machine learning models against deep learning models and (2) supervised learning models against unsupervised learning models. Our results suggest that the type of model (logistic regression, isolation forest, BIRCH, multi-layer perceptron, etc.) is more important than traditional machine learning vs. deep learning or supervised learning vs. unsupervised learning. Among our supervised traditional machine learning models, several performed very well (e.g., logistic regression, isolation forest); others performed

Table 15
G-mean. (Geometric mean of sensitivity & specificity.)

Model	#1, #1	#1, #2	#1, #3	#1, #4	#4, #1
Gaussian Naive Bayes	0.8848	0.8759	0.8846	0.8910	0.9861
K-Nearest Neighbors	0.5585	0.0100	0.0224	0.1216	0.4444
Linear Regression	0.3503	0.3050	0.3468	0.3272	0.3240
Logistic Regression	0.9927	0.9684	0.9987	0.9981	0.9891
Linear Support Vector Machine	0.5698	0.1884	0.0300	0.0299	0.0000
One-Class Support Vector Machine[a]	-	-	-	-	-
Support Vector Machine	0.5663	0.1887	0.0300	0.0245	0.3541
Decision Tree	0.3324	0.0000	0.1207	0.1648	0.6771
Extra Trees	0.5517	0.0000	0.0100	0.0000	0.4967
Gradient Boosting	0.6233	0.6582	0.9988	0.9989	0.9898
Isolation Forest	0.9901	0.9674	0.9980	0.9885	0.9881
Random Forest	0.0283	0.0000	0.0200	0.0100	0.5442
K-Means Clustering	0.8612	0.8766	0.8659	0.8873	0.7777
Mini-Batch K-Means Clustering	0.4705	0.5034	0.4736	0.4746	0.7675
BIRCH	0.9944	0.9872	0.9988	0.9995	0.9891
Local Outlier Factor	0.9298	0.1635	0.9393	0.1652	0.9556
Multi-Layer Perceptron	0.9824	0.9786	0.9986	0.9978	0.9898
Restricted Boltzmann Machine	0.0000	0.0000	0.0000	0.0000	0.1081

[a] The one-class support vector machine exceeded our 6-day (518400000000000 ns) time limit.

very poorly (e.g., support vector machine, extra trees). With unsupervised traditional machine learning, we saw exceptional performance from the BIRCH model, mediocre performance from mini-batch k-means clustering model, and inconsistent performance from the local outlier factor model. When it comes to deep learning, we have only two models to consider: (1) the supervised multi-layer perceptron and (2) the unsupervised restricted Boltzmann machine. The multi-layer perceptron performed exceptionally well, while the restricted Boltzmann machine performed exceptionally poorly. Therefore, we find both traditional machine learning and deep learning approaches to be well suited to the problem of automotive intrusion detection. Likewise, we find that both supervised and unsupervised models—if properly constructed and optimized—can function as automotive IDSs.

7.2.5. *Class imbalance*

An inherent challenge in intrusion detection is class imbalance, due to the fact that the vast majority of the network traffic is "normal," and the actual intrusions (i.e., attacks) represent an extremely rare minority. Intrusion detection falls under the umbrella of *rare event classification* problems, in which a machine learning model is expected to classify rare events (Brownlee, 2020). Such problems are, by nature, subject to class imbalance. As mentioned in Section 5.5.1, our `can-train-and-test` dataset is similarly imbalanced.

The accuracy metric is often ill-suited to imbalanced datasets (Brownlee, 2020; Kubat and Matwin, 1997); as such, in addition to accuracy, we have included the precision, recall, and F1-scores in Tables 10, 11, 12, 13, and 14. The recall metric is also known as the *sensitivity* metric or *true positive rate (TPR)*. Sensitivity is a popular choice when the positive class (in our case, the *attack* class) is extremely rare. When positive class is rare, a sub-standard classifier can assign all samples to the negative class and still achieve a high accuracy value. However, the sensitivity metric assesses how well the classifier identified the positive class. If the classifier assigned all samples to the negative class, its recall would be zero. Therefore, the sensitivity metric can be extremely useful to us as we examine the results of our imbalanced classification problem. In addition, we have the F1-score (also called the F-measure), which balances the concerns of precision and recall (i.e., sensitivity). A very high F1-score reassures us that our machine learning model is not too biased toward false negatives nor false positives (Brownlee, 2020; Guo et al., 2016).

In Tables 10, 11, 12, 13, and 14, we can see that four of our five top models—multi-layer perceptron, isolation forest, BIRCH, and logistic regression—achieved scores above 0.9 for accuracy, precision, recall, *and* F1-score. As such, we are assured of the capabilities of the models even when pitted against imbalanced datasets. The gradient boosting model, also in our top five, occasionally sees lower accuracy and recall score (between 0.6 and 0.7). The lower scores appeared when the gradient boosting model was pitted against sub-dataset #1, testing subsets #1 and #2. In sub-dataset #1, testing subset #1 has a class ratio (`attack-free:attack`) of 89:1. Testing subset #2 has a class ratio of 38:1. In contrast, testing subsets #3 and #4 have class ratios of 413:1 and 927:1, and we can see that the gradient boosting model performed much better against these two subsets. However, the gradient boosting model also performed well against sub-dataset #4, testing subset #1, where the class ratio was 44:1—closer to the ratios of testing subsets #1 and #2 in sub-dataset #1. As such, we can attribute some of the performance difference to additional factors (e.g., different vehicles, different scenarios). On the opposite side of the performance spectrum, we have the extra trees model. In sub-dataset #1, testing subset #1, if we just looked at accuracy (0.9915), or if we just looked at accuracy (0.9915) and precision (0.8199), we might believe the model is quite promising. But, when we look at recall (0.3046), we realize that the model is biased to underpredict the positive class, and the class imbalance is allowing the model to achieve reasonable accuracy and precision values. If we use the F1-score as our performance metric, then we can see that the extra trees model has some issues—the F1-score is 0.4442.

In Table 15, we provide the *g-mean*, that is, the geometric mean of sensitivity and specificity (Kubat and Matwin, 1997). The g-mean balances the concerns of sensitivity (i.e., recall or true positive rate) and specificity (true negative rate). Similar to the F1-score, the g-mean is a popular metric when dealing with class imbalance (Kubat and Matwin, 1997; Guo et al., 2016; Brownlee, 2020). As in our sensitivity analysis, we can see that four of our five top models—multi-layer perceptron, isolation forest, BIRCH, and logistic regression—achieved g-means above 0.9 for all five experiments. As before, the gradient boosting model struggled with sub-dataset #1, testing subsets #1 and #2. Having examined metrics that are well-suited to imbalanced datasets, we are reassured that the results of our benchmark are reliable and useful.

We have included detailed metrics—per model—in the appendix (see Appendix C). In our detailed metrics, we include raw numbers for true positives (TPs), true negatives (TNs), false positives (FPs), and false negatives (FNs). The raw numbers provide additional insights into the capabilities of each machine learning model as well as the impacts of class imbalance.

In addition, the `can-benchmark` repository contains the artifacts generated over the course of our benchmark evaluation. We also provide a Microsoft Excel spreadsheet, `benchmark.xlsx`, in which we meticulously organize the results of our experiments. `benchmark.xlsx` includes the F1-scores and g-means of all the machine learning models across all experiments, providing an excellent picture of the models' capabilities—in spite of the class imbalance. The `can-benchmark` repository is available here: https://bitbucket.org/brooke-lampe/can-benchmark/src/master/.

8. Limitations

In this section, we assess the limitations of our dataset. In particular, we revisit the limitations identified in Section 1.1:

1. Lack of sufficient attack-free data
2. Lack of sufficient attack variation
3. Lack of sufficient vehicle variation
4. Lack of fidelity
5. Lack of severity
6. Lack of modernity
7. Lack of labels

In addition, we compare our dataset to pre-existing open-access CAN bus datasets. Depending on the use case, some datasets are better suited to automotive research than ours.

Lack of sufficient attack-free data. We have addressed this limitation of existing datasets. Our dataset contains ample attack-free data. There are 91,827,504 total attack-free samples (i.e., captured CAN frames). There are *at least* 10,000,000 attack-free samples per vehicle.

Lack of sufficient attack variation. Nine unique attacks are represented in our dataset. Nonetheless, our attacks are not comprehensive. Our dataset lacks confliction-free masquerade attacks (all of our masquerade attacks involve confliction). If practitioners are looking for confliction-free masquerade attacks, then the Real ORNL Automotive Dynamometer CAN intrusion dataset (Verma et al., 2020, 2022) would be more suitable. In addition, our dataset lacks suppress attacks; as such, we refer practitioners to the TU Eindhoven CAN bus intrusion dataset v2 (Dupont et al., 2019) and the Synthetic CAN Bus dataset (Hanselmann et al., 2020), as they both provide suppress attacks. Lastly, our dataset lacks diagnostic attacks; to our knowledge, the TU CAN v2 (Dupont et al., 2019) contains the only publicly-available diagnostic attack. We plan to expand our dataset to include confliction-free masquerade attacks and suppress attacks in our future work (see Section 9).

Lack of sufficient vehicle variation. Four vehicle models and two vehicle manufacturers are represented in our dataset. To our knowledge, the greatest number of test vehicles in an existing open-access CAN bus dataset is three (HCRL Survival Analysis dataset (Han et al., 2018)). That said, the HCRL SA contains data from three different manufacturers (Chevrolet, Hyundai, Kia), whereas our dataset contains data from only two distinct manufacturers (Chevrolet, Subaru). In our future work, we plan to augment our dataset to represent more vehicle manufacturers and more vehicle models.

Lack of fidelity. We sought to curate a high-fidelity CAN bus intrusion dataset whenever possible, but, for safety reasons, some of our attacks were conducted offline. We replayed our captured CAN bus traffic and injected attacks using a combination of Linux's `SocketCAN` drivers (Community, 2023) and `can-utils` utilities (can-utils Contributors, 2023) and Python's `python-can` package (python-can Contributors, 2023b,a). The ROAD dataset (Verma et al., 2020, 2022) contains attacks that were conducted while the vehicle was in a dynamometer—i.e., a "rolling road" (de Menezes Lourenço et al., 2022). Such attacks would be more realistic than our offline attacks. That said, the CAN traffic data in the ROAD dataset has been obfuscated to protect the identity of the vehicle used for data collection; as such, there may be fidelity impacts. In particular, when the authors anonymized the arbitration IDs, they did not preserve the priority-ordering of the arbitration IDs.

Lack of severity. As we did not wish to damage the vehicle nor endanger the drivers and data collectors, we could not attempt the highest-severity attacks. We identified an attack that would put the vehicle in "neutral," and we conducted this attack—with a successful, observable result—both when the vehicle was actually in "drive" and when the vehicle was actually in "reverse." We theorize that spoofing "reverse" while in "drive" (or vice versa) using the same technique would also result in a successful, observable attack. However, if the attack were successful, it would strip the gears, damaging the vehicle and possibly endangering the occupants. As such, while we were able to conduct attacks that are more severe than changing the radio station, we were not able to conduct high-severity attacks that would jeopardize the vehicle or the occupants. This is a limitation of our dataset. The accelerator attacks in the ROAD dataset (Verma et al., 2020, 2022) are perhaps the most severe attacks actually conducted on a vehicle in motion.

Lack of modernity. Our dataset contains attacks that were customized to thwart the enhanced security of the newer vehicles we tested. We managed to conduct successful, observable attacks by transmitting a minimum number of spoofed messages in order to avoid being ejected from the CAN bus. That said, our newest vehicles were from 2016 and 2017. We cannot point practitioners to a specific dataset for more recent data, as we were unable to determine the model years for the vehicles in existing open-access datasets.

Lack of labels. We have addressed this limitation of existing datasets. Our `can-train-and-test` dataset, intended to support supervised learning, is completely labeled—"0" indicates an attack-free CAN frame, while "1" indicates an attack CAN frame.

Additional limitations. Compared to existing open-access CAN bus datasets, we have identified two additional limitations: (1) lack of signal data and (2) lack of scenario and positioning (i.e., location) information. We have not translated our raw CAN bus traffic data into discrete signals (i.e., discrete pieces of information). For signal data, practitioners should explore the AEGIS Big Data Project Automotive CAN Bus dataset (Kaiser et al., 2019), the Synthetic CAN Bus (SynCAN) dataset (Hanselmann et al., 2020), the HCRL CAN Signal Extraction and Translation dataset (Song and Kim, 2020), the Real ORNL Automotive Dynamometer CAN intrusion (ROAD) dataset (Verma et al., 2020, 2022), and the Reverse Engineering CAN Bus (ReCAN) dataset (Zago et al., 2020b,a). We do not provide information about specific driving scenarios (e.g., "driving at 40 mph, then lane change, then accelerate to 75 mph"), and we do not provide global positioning system (GPS) data. Practitioners seeking detailed driving information should refer to CrySyS Lab CAN dataset (CrySyS Lab, 2017), while practitioners interested in GPS data should explore AEGIS CAN (Kaiser et al., 2019).

9. Future work

We have identified a number of opportunities for future work, which are highlighted in Fig. 12. More specifically, we enumerate the following:

1. Craft advanced masquerade attacks (i.e., no confliction)
2. Dedicate experiments to optimization (i.e., finding the best parameters)
3. Construct additional models (e.g., non-learning, traditional machine learning, deep learning)
4. Collect diversified data (e.g., different vehicles, drivers, environments)
5. Evaluate our machine learning models against related works (e.g., other intrusion detection datasets, other machine learning models)
6. Conduct a fine-grained benchmark (e.g., which attacks are easiest for the decision tree model to detect?)

Advanced masquerade attacks. Our datasets contain unsophisticated masquerade attacks in which our malicious node "masquerades" as a legitimate ECU by using the legitimate ECU's arbitration identifier and transmitting spoofed CAN frames that resemble the legitimate ECU's messages. In this manner, we fooled the vehicle into responding

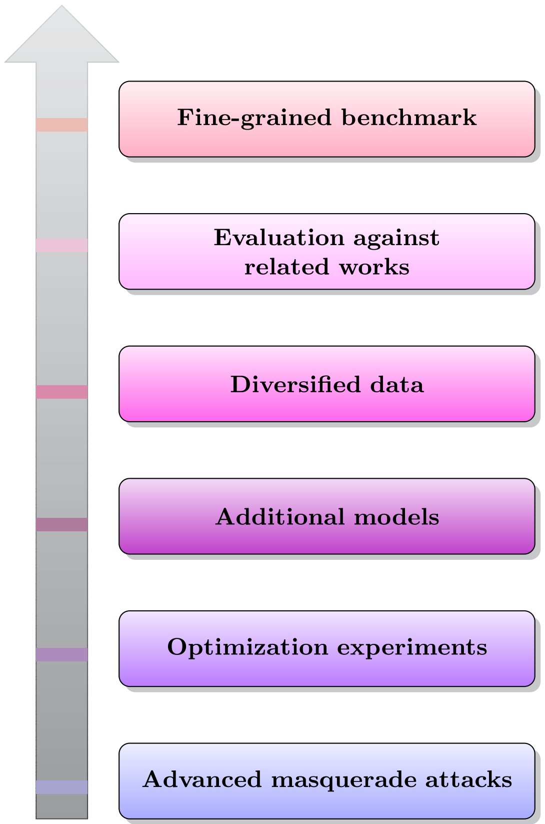

Fig. 12. Future work, prioritized.

to our malicious CAN frames as though they came from the legitimate ECU we were pretending to be. However, in our *unsophisticated* masquerade attack, there was still confliction. As mentioned in Section 2, a *sophisticated* masquerade attack incorporates both spoofing and suppress attacks. Essentially, CAN frames from the legitimate ECU are "suppressed," so that there is no confliction when the attacker "spoofs" CAN frames with the legitimate ECU's arbitration identifier. As such, the attacker "masquerades" as the legitimate ECU. To construct advanced masquerade attacks, we plan to explore two options: (1) scrubbing legitimate CAN frames from a traffic capture during post-processing or (2) conducting actual suppress attacks. It would be easier to simply remove legitimate CAN frames from a traffic capture during post-processing in order to simulate an advanced masquerade attack, but we are concerned about loss of fidelity. If we were to conduct an actual suppress attack in a live vehicle, fidelity would be preserved, but the test vehicle might be adversely impacted. Successful suppress attacks on live vehicles generally involve ECU reprogramming (Miller and Valasek, 2016b,a; Nie et al., 2017; Cai et al., 2019). If the ECU reprogramming process goes awry, we might not be able to restore the ECU to a functional state. Therefore, we will explore and evaluate our two options. Once we have suppressed the legitimate ECU's CAN frames—either live or during post-processing—we will conduct a spoofing attack. Because the legitimate ECU's CAN frames have been suppressed, there will be no confliction during the spoofing attack.

Optimization experiments. For our 18 machine learning models, we explored different parameter settings and conducted experiments with the goal of optimization. However, the focus of this work is our dataset and benchmark; as such, we did not rigorously optimize our models. In our future work, we plan to dedicate a number of experiments to optimization. We will meticulously record our parameters and results with the objective of finding optimal parameters for each model.

Additional models. In our future work, we plan to explore additional machine learning models—beyond `scikit-learn`'s offerings (Pedregosa et al., 2011). We will investigate both traditional machine learning and deep learning models, and we will experiment with both supervised and unsupervised learning paradigms. In particular, we are considering convolutional neural networks (CNNs) and transformers for our supervised deep learning models. For our unsupervised deep learning models, we are looking into autoencoders and generative adversarial networks (GANs) (Lampe and Meng, 2023c,b). We are also investigating the possibilities of long short-term memory (LSTM) networks, both for supervised and unsupervised learning (Lampe and Meng, 2023b). In addition, we intend to implement non-learning intrusion detection algorithms—e.g., interval-, frequency-, sequence-, and entropy-based techniques—to enrich our benchmark.

Diversified data. Automotive controller area network data varies widely by vehicle type, by manufacturer, and even by model. Though the 2011 Chevrolet Impala and the 2016 Chevrolet Silverado were both produced by General Motors and share the Chevrolet mark, they differ significantly when it comes to CAN frames. They share some arbitration identifiers but not others. If we look closely at the arbitration IDs they actually share, we can see that the data field is often longer for the Silverado's CAN frames than the Impala's. As such, it is essential to collect diversified data from different vehicle types, manufacturers, and models. A larger, more diversified dataset would provide a greater quantity of data—which is especially important for machine learning IDSs. In addition, we plan to recruit more test drivers. In particular, we will be looking at the experience level of our test drivers—inexperienced vs. experienced. We will also consider different test environments. Inter-city driving, for example, is drastically different from highway driving. Similarly, vehicles handle differently—and, thus, people drive differently—on minimum-maintenance gravel roads compared to paved asphalt roads. Lastly, we will look at additional driving conditions (e.g., weather) when enriching our dataset.

Evaluation against related works. The main contribution of this paper is our dataset. Our machine learning models (traditional machine learning, deep learning, supervised learning, unsupervised learning) were developed and evaluated to provide a "benchmark" for our dataset. Essentially, the benchmark demonstrates how different machine learning models might perform against our dataset. Moreover, the benchmark establishes a baseline for practitioners (i.e., researchers who use our dataset). They can evaluate the efficacy of their own techniques by comparing their techniques to our baseline. As such, as mentioned above, we did not rigorously optimize our machine learning models. Once we have optimized our models, we plan to pit them against the intrusion detection datasets made available in related works. We will compare our dataset to the datasets supplied by related works, and we will evaluate the efficacy of our machine learning models across different datasets. In addition, we will compare our machine learning models to the techniques described in related works. We plan to implement some of the techniques described in related works ourselves and evaluate them against our datasets.

Fine-grained benchmark. Once we have cultivated additional attacks, additional—optimized—models, and additional data, we plan to conduct a fine-grained benchmark. In this work, we have evaluated the performance of our models against our four train/test sub-datasets and our four testing subsets. In a more fine-grained benchmark, we will also consider the performance of a given model against an attack of a given type. We will answer questions such as...

1. Which attacks are easiest for the decision tree model to detect?
2. Which model is best at detecting speed spoofing attacks?
3. Across all models, which attacks are easiest to detect? Hardest?

10. Conclusion

The controller area network (CAN) bus has emerged as the de facto standard for in-vehicle networks (IVNs) around the globe. Safety-critical components (e.g., the brakes, the engine, the transmission) depend on the CAN bus for expedient, reliable communication. Unfortunately, while the CAN bus was designed to be resilient under harsh operating conditions, it was not designed to be resilient under *adversarial* conditions. Standard security practices such as authentication, authorization, and encryption are completely lacking when it comes to the CAN bus. Researchers have since developed authentication, authorization, and encryption specifications for the CAN bus, but retroactive implementation of said security controls would be exorbitantly expensive—in terms of hardware, labor, engineering effort, and monetary cost. Therefore, the automotive intrusion detection system (IDS) has emerged in the literature as a low-cost, low-effort solution to the automotive [in]security problem. However, developing and evaluating an automotive IDS can be quite challenging—especially if researchers lack access to a test vehicle. Without a test vehicle, researchers are limited to publicly available CAN data, and existing CAN intrusion detection datasets come with various limitations. This lack of CAN data has become a barrier to entry into automotive intrusion detection research—and even automotive security research in general.

We seek to lower this barrier to entry by introducing a new CAN intrusion detection dataset, which facilitates the development and evaluation of automotive IDSs. Our dataset, `can-train-and-test`, offers real-world CAN traffic data from four different vehicles—a sedan, a compact SUV, a full-size SUV, and a pickup truck—produced by two different manufacturers. For each vehicle, we provided comparable attack captures, which enable researchers to assess a given IDS's ability to generalize to different vehicle types and models. Our dataset contains `.log` files for playback as well as labeled and unlabeled `.csv` files for supervised and unsupervised machine learning. As such, our dataset is well suited to a variety of different automotive intrusion detection and automotive security enterprises. In addition, `can-train-and-test` supplies nine unique attacks, ranging from denial of service (DoS) fuzzing to triple spoofing attacks. As such, researchers can select from a wide variety of attacks when partitioning the data into training and testing datasets. Alternatively, researchers can leverage our curated `can-train-and-test` repository, which is subdivided into four train/test sub-datasets and four testing subsets. As a benchmark, we pitted 18 machine learning models against the `can-train-and-test` repository. During our evaluation and analysis, we found that the multi-layer perceptron, gradient boosting, isolation forest, BIRCH, and logistic regression models consistently scored above 0.95 when it came to accuracy, precision, recall, and F1-score—regardless of the sub-dataset and testing subset. Across all experiments on all sub-datasets, we saw an average F1-score of ≈0.5039, indicating that our `can-train-and-test` dataset is indeed capable of distinguishing capable, well-trained IDSs from their less-than-capable counterparts. We present `can-train-and-test` as a contribution to the existing collection of open-access CAN intrusion detection datasets in hopes of filling in the gaps left by the existing collection.

CRediT authorship contribution statement

Brooke Lampe: Conceptualization, Data curation, Investigation, Methodology, Writing – original draft. **Weizhi Meng:** Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared the data links in our paper.

Acknowledgements

We would like to thank our test drivers, especially Christian Lampe, Julie Lampe, and Wayne & Vickie Olsen.

Appendix A. CAN traffic samples

A.1. Attack-free

Listings 13 and 14 showcase attack-free traffic in driving mode and accessory mode, respectively. The traffic was collected from the 2011 Chevrolet Traverse.

Listing 13 is the raw, unlabeled equivalent of 23 in Section Appendix B.1.

A.2. Denial of Service (DoS)

Listings 15 and 16 both showcase Denial of Service (DoS) attacks. In Listing 15, the DoS attack is extremely overt (the invalid arbitration ID `000` is used), whereas in Listing 16, the DoS attack is more subtle (the valid arbitration ID `0C1` is used). The traffic was collected from the 2011 Chevrolet Traverse.

A.3. Fuzzing

Listing 17 demonstrates a fuzzing attack. The traffic was collected from the 2011 Chevrolet Traverse.

This is the raw, unlabeled equivalent of 24 in Section Appendix B.2.

A.4. Spoofing

Listings 18 and 19 showcase spoofing attacks—a gear spoofing attack and a triple spoofing attack, respectively. In the gear spoofing attack, the "neutral" gear is spoofed; the true value is "drive." In the triple spoofing attack, the gear, the RPMs, and the speed are all spoofed (arbitration IDs `1F5`, `0C9`, `3E9`, respectively). The vehicle is driving at speed, but the spoofed messages indicate that it is in "neutral" with no acceleration and negligible speed. The traffic was collected from the 2011 Chevrolet Traverse.

A.5. Interval

Listing 20 demonstrates an interval attack, in which the target arbitration ID appears on the CAN bus, and, immediately, the adversary transmits a spoofed message to countermand the true message. Depending on bus traffic and arbitration, the spoofed message may directly follow the true message, or there may be a few messages in between. The traffic was collected from the 2011 Chevrolet Traverse.

A.6. Standstill

Listing 21 outlines a standstill attack, in which the vehicle is tricked into behaving as though in "neutral." The traffic was collected from the 2011 Chevrolet Traverse.

A.7. Systematic

Listing 22 spotlights a systematic attack, which might be used for reconnaissance. The traffic was collected from the 2011 Chevrolet Traverse.

Listing 13 Attack-free traffic (driving mode).

```
(1672163108.460031) can0 1E1#0000100000
(1672163108.460701) can0 0C7#00D4A352
(1672163108.460932) can0 0F9#031A4002A23FDEFA
(1672163108.461181) can0 189#4FFF0FFF3000DEFA
(1672163108.461405) can0 199#4FFF0E70F18F00FF
(1672163108.461548) can0 1EB#0141
(1672163108.462257) can0 1F1#AE0E00000800007A
(1672163108.462765) can0 0C1#236F7087236A5491
(1672163108.462912) can0 1CB#10E600
(1672163108.463124) can0 0C5#234188A42341E475
(1672163108.463359) can0 184#0001000001FF
(1672163108.463584) can0 1C7#0FFF700003FF3F
(1672163108.463758) can0 1CD#C7FF07FE7F
(1672163108.464028) can0 1E5#4600D8EED4FF2500
(1672163108.464229) can0 1E9#0FFF000C00260000
(1672163108.464477) can0 0C9#801D551911000000
(1672163108.464634) can0 0F1#1C040040
(1672163108.464846) can0 191#075E075E0761115D
(1672163108.465105) can0 1ED#41300809067B0800
(1672163108.465277) can0 1EF#00000920
(1672163108.465558) can0 2C3#0905066906697200
(1672163108.465669) can0 2F9#80010F002C
(1672163108.465829) can0 334#0000
(1672163108.466006) can0 348#030B0306
(1672163108.466251) can0 17D#04E000007D000100
(1672163108.466551) can0 17F#0000000000000000
(1672163108.466708) can0 34A#03110306
(1672163108.467128) can0 1F3#0020
(1672163108.469324) can0 3C9#0766000000000000
(1672163108.472589) can0 0C1#33717FE0336B5C56
(1672163108.472812) can0 0C5#334398013342EC32
(1672163108.473036) can0 1E5#4600D48E70FF2C00
(1672163108.473210) can0 0C7#00DCAFFE
(1672163108.473445) can0 0F9#031A4002A23FDEFA
(1672163108.473690) can0 189#8FFF0FFF2FFFDEFA
(1672163108.473812) can0 1EB#0124
(1672163108.474059) can0 199#8FFF0E70F18E00FF
(1672163108.474398) can0 0C9#801D4E1C10000000
(1672163108.474590) can0 0F1#28040040
(1672163108.474812) can0 191#075A075A075E105A
(1672163108.475034) can0 1ED#413007ED06730800
(1672163108.475195) can0 1EF#000008BE
(1672163108.475455) can0 1A1#0000414000001000
(1672163108.475689) can0 1C3#075A076A00000000
(1672163108.481761) can0 19D#40003FFF0018105C
(1672163108.481922) can0 1AF#000000
(1672163108.482180) can0 1F5#0202000400000900
(1672163108.482760) can0 0C1#03728785036D6BE1
(1672163108.482907) can0 1CB#10D900
(1672163108.483118) can0 0C5#03449FB80344FBAA
```

Listing 14 Attack free traffic (accessory mode).

```
(1672176421.032178) can0 19D#C0003FFD000000FF
(1672176421.032397) can0 1C7#0FFF300103FF3F
(1672176421.032548) can0 1AF#000000
(1672176421.032732) can0 1CD#87FF07FF7F
(1672176421.032981) can0 1E5#46FFC2A000003D00
(1672176421.033226) can0 1E9#0000000C00000000
(1672176421.033477) can0 1ED#0000000000000800
(1672176421.033625) can0 1EF#0000002F
(1672176421.033887) can0 1F5#0F0F000100000300
(1672176421.034132) can0 3C1#0765040000000000
(1672176421.034383) can0 3D1#1046000000000000
(1672176421.034632) can0 3E9#0000000000000000
(1672176421.035391) can0 0F1#1C050040
(1672176421.035959) can0 0C7#03FE0000
(1672176421.036231) can0 0F9#00004000000000FF
(1672176421.036505) can0 189#CFFF0FFF2FFE00FF
(1672176421.036735) can0 199#CFFF0E70F18D00FF
(1672176421.037429) can0 1EB#008A
(1672176421.038562) can0 1F3#C0E0
(1672176421.040835) can0 0C1#1000000010000000
(1672176421.041107) can0 0C5#1000000010000000
(1672176421.041350) can0 0C9#0000000D00000000
(1672176421.041577) can0 191#063D091E091E0000
(1672176421.041812) can0 1E5#46FFC2C000003C00
(1672176421.042055) can0 1ED#0000000000000800
(1672176421.042300) can0 1A1#0010010000000000
(1672176421.042552) can0 1C3#063D063D00000000
(1672176421.042732) can0 1EF#0000002F
(1672176421.042826) can0 334#0000
(1672176421.045354) can0 0F1#28050040
(1672176421.045657) can0 1CB#100000
(1672176421.047933) can0 12A#47000000C2563FFF
(1672176421.048535) can0 0C7#03FE0000
(1672176421.048805) can0 0F9#00004000000000FF
(1672176421.049056) can0 189#0FFF0FFF300100FF
(1672176421.049407) can0 199#0FFF0E70F19000FF
(1672176421.049817) can0 1EB#008A
(1672176421.050556) can0 1E1#00FE120040
(1672176421.051037) can0 0C1#2000000020000000
(1672176421.051281) can0 0C9#0000000000000000
(1672176421.051570) can0 0C5#2000000020000000
(1672176421.051728) can0 184#0001000001FF
(1672176421.051958) can0 191#063D091E091E0000
(1672176421.052177) can0 1C7#0FFF700003FF3F
(1672176421.052418) can0 1CD#C7FF07FE7F
(1672176421.052625) can0 1E5#46FFC2E000003B00
(1672176421.052863) can0 1E9#0000000C00000000
(1672176421.053111) can0 1ED#0000000000000800
(1672176421.053283) can0 1EF#0000002F
(1672176421.053528) can0 1F1#AE0FBB140800007A
```

Listing 15 An overt Denial of Service (DoS) attack. The red lines indicate attack CAN frames.

```
(1672451549.617791) can0 17D#04E000007D000100
(1672451549.618710) can0 000#0000000000000000
(1672451549.618856) can0 17F#0000000000000000
(1672451549.618865) can0 34A#00370037
(1672451549.618872) can0 3ED#8000000000FF0000
(1672451549.619939) can0 19D#000000000002BB8F
(1672451549.619948) can0 1AF#000000
(1672451549.619955) can0 1F5#0101000400000900
(1672451549.620107) can0 000#0000000000000000
(1672451549.621021) can0 0C9#840BD00D00000000
(1672451549.621031) can0 191#06CD06DC06CE0000
(1672451549.621309) can0 000#0000000000000000
(1672451549.622181) can0 1ED#413E018E01830800
(1672451549.622199) can0 1EF#00000212
(1672451549.622205) can0 1A1#0000414000000000
(1672451549.622221) can0 1C3#06CD06CE00000000
(1672451549.622856) can0 000#0000000000000000
(1672451549.623303) can0 0C7#03F2CB83
(1672451549.624270) can0 000#0000000000000000
(1672451549.624394) can0 0F9#070F400031E5CAFF
(1672451549.624410) can0 189#4FFF0FFF3000CAFF
(1672451549.624417) can0 199#4FFF0E70F18F00FF
(1672451549.625397) can0 000#0000000000000000
(1672451549.625597) can0 0F1#28050040
(1672451549.625622) can0 1E1#00FE120040
(1672451549.625629) can0 0C1#30050E3F3318FD90
(1672451549.626313) can0 000#0000000000000000
(1672451549.626704) can0 0C5#32BC782532768E91
(1672451549.626714) can0 1E5#46FFF58000000B00
(1672451549.626720) can0 1EB#0144
(1672451549.626938) can0 000#0000000000000000
(1672451549.627810) can0 000#0000000000000000
(1672451549.628514) can0 000#0000000000000000
(1672451549.629338) can0 000#0000000000000000
(1672451549.630615) can0 000#0000000000000000
(1672451549.630962) can0 0C9#840BD30000000000
(1672451549.631843) can0 000#0000000000000000
(1672451549.632021) can0 191#06CD06DC06CE0000
(1672451549.632025) can0 1CB#100000
(1672451549.632028) can0 1ED#413E018E01830800
(1672451549.632030) can0 1EF#00000215
(1672451549.632032) can0 2D1#030000000000
(1672451549.632034) can0 3FD#003C3D
(1672451549.633086) can0 000#0000000000000000
(1672451549.634342) can0 000#0000000000000000
(1672451549.635207) can0 0F1#34050040
(1672451549.635233) can0 000#0000000000000000
(1672451549.636044) can0 000#0000000000000000
(1672451549.636271) can0 0C1#00050E3F0318FD90
(1672451549.636276) can0 0C7#03F3E107
```

Listing 16 A subtle Denial of Service (DoS) attack. The red lines indicate attack CAN frames.

```
(1672452653.697771) can0 3C9#0766000000000000
(1672452653.698848) can0 1EB#012C
(1672452653.698907) can0 0C1#23F612A62309F450
(1672452653.700027) can0 0C1#23F612A62309F450
(1672452653.700934) can0 0C1#23F612A62309F450
(1672452653.701712) can0 0C1#23F612A62309F450
(1672452653.702278) can0 0C1#23F612A62309F450
(1672452653.703328) can0 0C1#23F612A62309F450
(1672452653.704221) can0 0C9#840AFE0000010000
(1672452653.704226) can0 191#06B606C306B50000
(1672452653.704226) can0 0C1#23F612A62309F450
(1672452653.704229) can0 1ED#413B018301710800
(1672452653.705285) can0 1EF#000001EE
(1672452653.705288) can0 0C1#22F94EA9228F2EA9
(1672452653.705471) can0 0C1#23F612A62309F450
(1672452653.706344) can0 0C5#22CB6027225F289B
(1672452653.706348) can0 184#0003002C01D1
(1672452653.706350) can0 0C7#020AC94F
(1672452653.706353) can0 1C7#0FFFEFFE03FF3F
(1672452653.706355) can0 0F9#07CD4000001D60FF
(1672452653.706614) can0 0C1#23F612A62309F450
(1672452653.707365) can0 0C1#23F612A62309F450
(1672452653.707437) can0 189#8FFF0FFF2FFF60FF
(1672452653.707454) can0 1CD#47FF08007F
(1672452653.707474) can0 199#8FFF0E70F18E00FF
(1672452653.707479) can0 1E5#46FFDCE000002100
(1672452653.707481) can0 0F1#225100C0
(1672452653.708523) can0 0C1#23F612A62309F450
(1672452653.708548) can0 1E9#4001000C00000000
(1672452653.708557) can0 2F9#E0010F0000
(1672452653.708564) can0 348#00000000
(1672452653.708566) can0 17D#04E000007D000100
(1672452653.708568) can0 17F#0000000000000000
(1672452653.709631) can0 34A#00000000
(1672452653.709665) can0 0C1#23F612A62309F450
(1672452653.710452) can0 0C1#23F612A62309F450
(1672452653.710695) can0 1E1#00FE120040
(1672452653.710700) can0 1EB#0130
(1672452653.711231) can0 0C1#23F612A62309F450
(1672452653.712007) can0 0C1#23F612A62309F450
(1672452653.712832) can0 0C1#23F612A62309F450
(1672452653.713775) can0 0C1#23F612A62309F450
(1672452653.713875) can0 0C9#840AFE0700010000
(1672452653.714407) can0 0C1#23F612A62309F450
(1672452653.714961) can0 191#06B606C306B50000
(1672452653.714986) can0 19D#40003FFF000000FF
(1672452653.714996) can0 1ED#413B018301710800
(1672452653.715005) can0 1AF#000000
(1672452653.715013) can0 1A1#0010414000000000
(1672452653.715441) can0 0C1#23F612A62309F450
```

Listing 17 A fuzzing attack. The red lines indicate attack CAN frames.

```
(1672452917.316296) can0 348#00000000
(1672452917.316301) can0 34A#00000000
(1672452917.317073) can0 036#F237
(1672452917.318151) can0 0EE#750F037DB0973413
(1672452917.318595) can0 0F1#34050040
(1672452917.319220) can0 63C#BB98131FFB6BB64D
(1672452917.319747) can0 19D#C0003FFD000000FF
(1672452917.319763) can0 1AF#000000
(1672452917.320301) can0 758#B4E4B84DB744242D
(1672452917.320835) can0 1F5#0F0F002100000300
(1672452917.320848) can0 1EB#0000
(1672452917.321445) can0 1B9#619E732063822815
(1672452917.322642) can0 107#A60ED87316918238
(1672452917.323068) can0 0C1#0000000000000000
(1672452917.323789) can0 3AA#9442CD7C6E7A727F
(1672452917.324170) can0 0C5#000000000002A384
(1672452917.324188) can0 184#0002000001FE
(1672452917.324193) can0 0C9#0000000A00000000
(1672452917.324197) can0 191#061408F508F50000
(1672452917.324202) can0 1C7#0FFFAFFF03FF3F
(1672452917.324865) can0 255#BA661E
(1672452917.325345) can0 1CD#07FF08017F
(1672452917.325361) can0 1E5#46FFF5A000000A00
(1672452917.325366) can0 1E9#0000000C00000000
(1672452917.325370) can0 0C7#03FE0000
(1672452917.325374) can0 1ED#0000000000000800
(1672452917.326023) can0 665#E9ECB608535FB713
(1672452917.326522) can0 0F9#00004000000000FF
(1672452917.326540) can0 1EF#00000000
(1672452917.326547) can0 189#CFFF0FFF2FFE00FF
(1672452917.326552) can0 334#0000
(1672452917.326558) can0 199#CFFF0E70F18D00FF
(1672452917.327177) can0 218#03F7EB2654A8732F
(1672452917.327708) can0 0F1#00050040
(1672452917.327718) can0 1F3#8080
(1672452917.328247) can0 5A8#4F142A7D00B5B738
(1672452917.329390) can0 5DA#B8F9DB65
(1672452917.330561) can0 291#F6F9003C98C3F40E
(1672452917.331705) can0 6E7#AE54774792E2CD68
(1672452917.332127) can0 1CB#100000
(1672452917.332776) can0 335#005D
(1672452917.333326) can0 0C1#1000000010000000
(1672452917.333361) can0 0C5#100000001002A384
(1672452917.333376) can0 1E5#46FFF5C000000900
(1672452917.333944) can0 3A5#17E83D560AA2C903
(1672452917.334517) can0 1EB#0000
(1672452917.334528) can0 0C9#0000000D00000000
(1672452917.334531) can0 191#061408F508F50000
(1672452917.334533) can0 1ED#0000000000000800
(1672452917.334535) can0 1EF#00000000
```

Listing 18 A gear spoofing attack. The red lines indicate attack CAN frames.

```
(1674135810.232265) can0 17F#0000000000000000
(1674135810.232272) can0 34A#08D108CA
(1674135810.234101) can0 1F5#0D0D000300000300
(1674135810.235464) can0 1EB#0144
(1674135810.236563) can0 0C9#8025E6193B000000
(1674135810.236578) can0 191#07C407C407C43BAF
(1674135810.236583) can0 1ED#412F0B970C140800
(1674135810.237649) can0 1EF#00001119
(1674135810.237659) can0 2C3#0949066406648100
(1674135810.237666) can0 0C7#01B4671C
(1674135810.237676) can0 0F9#016E4007A457BB12
(1674135810.238748) can0 189#8FFF0FFF2FFFBB12
(1674135810.238758) can0 0F1#00060040
(1674135810.238767) can0 199#8FFF0E70F18E00FF
(1674135810.239824) can0 0C1#11DBBB58123AAF44
(1674135810.239829) can0 0C5#11805719120CB19E
(1674135810.239834) can0 1E5#46FFE0C000001E00
(1674135810.245141) can0 1CB#10E800
(1674135810.245806) can0 1F5#0D0D000300000300
(1674135810.246224) can0 19D#40003FFF0022272D
(1674135810.246235) can0 1AF#000010
(1674135810.247305) can0 0C9#8025F91C3B000000
(1674135810.247323) can0 191#07C407C407C43BAF
(1674135810.247331) can0 1ED#412F0BAB0C1C0800
(1674135810.247340) can0 1EF#00001100
(1674135810.247352) can0 1A1#0000414000003B00
(1674135810.247361) can0 1C3#07C407C000000000
(1674135810.248427) can0 1F5#2404000400000900
(1674135810.248436) can0 0F1#1C060040
(1674135810.248447) can0 1E1#00FD130060
(1674135810.248460) can0 1EB#010C
(1674135810.249532) can0 0C1#21DFC610223EBA00
(1674135810.249543) can0 0C5#218461D72210BC5B
(1674135810.250613) can0 0C7#01CA7333
(1674135810.250623) can0 0F9#016E4007A457D512
(1674135810.250635) can0 184#0003000001FD
(1674135810.250641) can0 189#CFFF0FFF2FFED512
(1674135810.250643) can0 1C7#0FFFEFFE03FF3F
(1674135810.251700) can0 199#CFFF0E70F18D00FF
(1674135810.251704) can0 1CD#47FF08007F
(1674135810.251707) can0 1E5#46FFE0E000001D00
(1674135810.251711) can0 1E9#0020000C00030000
(1674135810.257020) can0 0C9#8025FE133B000000
(1674135810.257036) can0 191#07C407C407C43BAE
(1674135810.257045) can0 1ED#412F0BAB0C1C0800
(1674135810.257049) can0 1EF#00001119
(1674135810.257648) can0 1F5#0D0D000300000300
(1674135810.258110) can0 0F1#28060040
(1674135810.258116) can0 1F3#80A0
(1674135810.259165) can0 0C1#31E3D0C83241C20A
```

Listing 19 A triple spoofing attack. The red lines indicate attack CAN frames.

```
(1674122022.785040) can0 1ED#412E019E01EC0733
(1674122022.785263) can0 3E9#0000000100000000
(1674122022.786106) can0 1EF#0000023C
(1674122022.786117) can0 1A1#0000414000000000
(1674122022.786120) can0 1C3#06C406C700000000
(1674122022.786193) can0 0C9#00000000000018
(1674122022.787200) can0 2C3#084F06C706C74A00
(1674122022.787216) can0 0C1#3293ED4A32C132FF
(1674122022.787221) can0 0C5#32513FBB328FF7ED
(1674122022.787226) can0 1E5#4607768514F88A00
(1674122022.787338) can0 1F5#0D0D000300000800
(1674122022.788295) can0 2F9#0B0111001D
(1674122022.788308) can0 348#01AB0187
(1674122022.788312) can0 34A#01A40187
(1674122022.788316) can0 0F1#00070040
(1674122022.790044) can0 3E9#0000000100000000
(1674122022.790451) can0 0C7#003F7994
(1674122022.790468) can0 0F9#00004001647A2EFA
(1674122022.790478) can0 189#8FFF0FFF2FFF2EFA
(1674122022.790482) can0 199#8FFF0E70F18E00FF
(1674122022.790896) can0 0C9#00000000000018
(1674122022.791559) can0 1EB#0135
(1674122022.791897) can0 1F5#0D0D000300000800
(1674122022.794745) can0 0C9#840B790000000000
(1674122022.794752) can0 191#06C606E506C70000
(1674122022.795123) can0 3E9#0000000100000000
(1674122022.795812) can0 1ED#412E019C01EB0733
(1674122022.795818) can0 1EF#0000023C
(1674122022.795821) can0 3C1#0765E70000000000
(1674122022.795826) can0 3D1#0122000000570000
(1674122022.795833) can0 1CB#101700
(1674122022.795841) can0 0C9#00000000000018
(1674122022.796598) can0 1F5#0D0D000300000800
(1674122022.796898) can0 3E9#035118F0034F18F9
(1674122022.796904) can0 0C1#0294FB5702C24242
(1674122022.797977) can0 0C5#02524E0402900732
(1674122022.797986) can0 184#000000000000
(1674122022.797989) can0 0F1#1C070040
(1674122022.797992) can0 1C7#0FFF300103FF3F
(1674122022.797995) can0 1CD#87FF07FF7F
(1674122022.799062) can0 19D#C0003FFD000CAC5C
(1674122022.799074) can0 1E5#460781A44CF87E00
(1674122022.799082) can0 1AF#000000
(1674122022.799085) can0 1E9#0FD9000C00800000
(1674122022.799093) can0 1F5#0202000400000900
(1674122022.799549) can0 3E9#0000000100000000
(1674122022.800162) can0 334#0000
(1674122022.800303) can0 0C9#00000000000018
(1674122022.801232) can0 1F3#4060
(1674122022.801282) can0 1F5#0D0D000300000800
```

Listing 20 An interval attack. The red lines indicate attack CAN frames.

```
(1674033486.438066) can0 1CD#C7FF07FE7F
(1674033486.438074) can0 1E5#460081E000FF7C00
(1674033486.438079) can0 1E9#4FFB000C00000000
(1674033486.438933) can0 1E9#000A000C00060000
(1674033486.440201) can0 1CB#100000
(1674033486.440212) can0 2D1#030000000000
(1674033486.441283) can0 3FD#003A3A
(1674033486.441294) can0 0F1#7E7F00C0
(1674033486.442361) can0 0C9#840AE80A00010000
(1674033486.442372) can0 191#06AB06AB06AB0000
(1674033486.442379) can0 1ED#418A019601B70800
(1674033486.442383) can0 1EF#0000026E
(1674033486.443457) can0 19D#80003FFE000A8AFF
(1674033486.443471) can0 1AF#000000
(1674033486.443480) can0 1F5#0F0F000100000800
(1674033486.444540) can0 1F3#4060
(1674033486.444547) can0 1EB#0199
(1674033486.445630) can0 4D9#000000
(1674033486.446699) can0 0C1#3000000030000000
(1674033486.447770) can0 0C5#3000000030000000
(1674033486.447786) can0 1E5#4600818000FF7F00
(1674033486.447790) can0 334#0000
(1674033486.447794) can0 0C7#03FE0000
(1674033486.447798) can0 0F9#00004000000000FF
(1674033486.448877) can0 189#4FFF0FFF300000FF
(1674033486.448886) can0 199#4FFF0E70F18F00FF
(1674033486.452073) can0 0C9#840ACE0D00010000
(1674033486.452088) can0 0F1#4A8000C0
(1674033486.452099) can0 191#06AD06AD06AB0000
(1674033486.452103) can0 1ED#4189019E01B40800
(1674033486.453170) can0 1EF#00000284
(1674033486.453186) can0 1A1#0020014000000000
(1674033486.453199) can0 1C3#06AD06B100000000
(1674033486.454265) can0 2C3#0800069806984C00
(1674033486.454280) can0 1E1#0000100000
(1674033486.457455) can0 0C1#0000000000000000
(1674033486.457481) can0 0C5#0000000000000000
(1674033486.457483) can0 184#00020080017E
(1674033486.457486) can0 1C7#0FFFAFFF03FF3F
(1674033486.457491) can0 1CD#07FF08017F
(1674033486.458560) can0 1E5#460081A000FF7E00
(1674033486.458575) can0 1E9#4FFA000C00000000
(1674033486.458596) can0 1EB#019B
(1674033486.459399) can0 1E9#000A000C00060000
(1674033486.460715) can0 1CB#100000
(1674033486.460729) can0 0C7#03FE0000
(1674033486.460732) can0 0F9#00004000000000FF
(1674033486.460736) can0 189#8FFF0FFF2FFF00FF
(1674033486.461803) can0 199#8FFF0E70F18E00FF
(1674033486.461817) can0 0F1#568201C0
```

Listing 21 A standstill attack. The red lines indicate attack CAN frames.

```
(1674135909.661269) can0 1EF#00000252
(1674135909.661280) can0 3C9#0766000000000000
(1674135909.661382) can0 0C9#0000000000004008
(1674135909.662355) can0 1EB#0151
(1674135909.662365) can0 0C7#02A02F6D
(1674135909.663433) can0 0F9#01004006A7ACCF1C
(1674135909.663444) can0 0F1#3E4300C0
(1674135909.663451) can0 189#CFFF0FFF2FFECF1C
(1674135909.663465) can0 19D#C0003FFD0013F31E
(1674135909.663474) can0 0C1#234941D123048C3C
(1674135909.664563) can0 0C5#22C2F18822B99C76
(1674135909.664575) can0 184#0003001801E5
(1674135909.664587) can0 199#CFFF0E70F18D00FF
(1674135909.664599) can0 1AF#000034
(1674135909.664604) can0 1C7#0FFFEFFE03FF3F
(1674135909.664607) can0 1CD#47FF08007F
(1674135909.665677) can0 1E5#46FFF8E000000500
(1674135909.665693) can0 1E9#0FFD000C00000000
(1674135909.665705) can0 1F5#6625000400000900
(1674135909.669959) can0 1CB#100000
(1674135909.669970) can0 0C9#8413F80700010000
(1674135909.669985) can0 191#063406A306340000
(1674135909.671051) can0 1ED#4144000001AC0860
(1674135909.671061) can0 1EF#00000255
(1674135909.671069) can0 1A1#0000414000000000
(1674135909.671079) can0 1C3#0634063500000000
(1674135909.672144) can0 2C3#0897063506353600
(1674135909.673221) can0 0F1#0A4300C0
(1674135909.673231) can0 0C1#334C4B0F33079579
(1674135909.673256) can0 0C9#0000000000004008
(1674135909.674302) can0 0C5#32C6FDDC32BCA5B3
(1674135909.674314) can0 1E5#46FFF88000000800
(1674135909.674319) can0 334#0000
(1674135909.674322) can0 1EB#0150
(1674135909.675389) can0 0C7#02B33B76
(1674135909.675399) can0 0F9#01004006A7ACCF1C
(1674135909.675407) can0 189#0FFF0FFF3001CF1C
(1674135909.676476) can0 199#0FFF0E70F19000FF
(1674135909.676486) can0 1E1#00FD130060
(1674135909.679669) can0 2D1#030000000000
(1674135909.680728) can0 0C9#8414180A00010000
(1674135909.680733) can0 191#063406A306340000
(1674135909.680735) can0 1ED#4144000001AE0860
(1674135909.680738) can0 1EF#00000258
(1674135909.680741) can0 3C1#0765E70000000000
(1674135909.681802) can0 3D1#0120400000000000
(1674135909.681806) can0 3E9#0F9A071B0F9D0724
(1674135909.682992) can0 0F1#1641FFC0
(1674135909.684085) can0 0C1#03505777030A9EC1
(1674135909.684095) can0 0C5#02C9071902C0B200
```

Listing 22 A systematic attack. The red lines indicate attack CAN frames.

```
(1672454739.897283) can0 1E5#46FFEFE000000E00
(1672454739.898351) can0 1E9#001D000C00030000
(1672454739.899311) can0 000#98BF3D7C9A5CC42F
(1672454739.899415) can0 0C7#018CBBB7
(1672454739.900398) can0 001#207215535845D22C
(1672454739.900485) can0 0F9#022340041C15FC0D
(1672454739.900495) can0 189#0FFF0FFF3001FC0D
(1672454739.900505) can0 199#0FFF0E70F19000FF
(1672454739.901544) can0 002#9C8B734717464C1F
(1672454739.901571) can0 0C9#8020C91337000000
(1672454739.901579) can0 191#07E807E807E837BC
(1672454739.901588) can0 1ED#412F0CB50B9D0800
(1672454739.901592) can0 1EF#000010C3
(1672454739.902660) can0 3F9#0012573257695414
(1672454739.902670) can0 1EB#011D
(1672454739.902680) can0 3FB#8100
(1672454739.902688) can0 003#3473696411F5
(1672454739.903746) can0 1CB#101E00
(1672454739.903869) can0 004#CB0ACB3A50163B23
(1672454739.905008) can0 005#BEA73D6DAAB703
(1672454739.905879) can0 0C1#300B5A29300496ED
(1672454739.906159) can0 006#72F8D77A3FF7DF72
(1672454739.906950) can0 0C5#33EE9A1B33E572B8
(1672454739.906960) can0 0F1#00050040
(1672454739.906969) can0 1E5#46FFEF8000001100
(1672454739.906975) can0 334#0000
(1672454739.907291) can0 007#0173523948F1693B
(1672454739.908041) can0 19D#000000000017FE3B
(1672454739.908085) can0 1AF#000000
(1672454739.908359) can0 008#82BB4A
(1672454739.909151) can0 1F5#0303000400000900
(1672454739.909164) can0 12A#47000000B2753FFF
(1672454739.909526) can0 009#7E66BA4837E38747
(1672454739.910613) can0 00A#BA8D9758CFA2C543
(1672454739.911303) can0 0C9#8020CE1637000000
(1672454739.911314) can0 191#07E707E707E737BC
(1672454739.911708) can0 00B#82D94D6AEF14DB16
(1672454739.912381) can0 1E1#0000100000
(1672454739.912392) can0 1ED#412F0CB50B9D0800
(1672454739.912396) can0 1EF#000010CE
(1672454739.912400) can0 0C7#0198C7F2
(1672454739.912404) can0 1A1#0000414000003700
(1672454739.912408) can0 0F9#022140041D160B0D
(1672454739.912784) can0 00C#ED
(1672454739.913474) can0 1C3#07E707D300000000
(1672454739.913484) can0 189#4FFF0FFF30000B0D
(1672454739.913488) can0 199#4FFF0E70F18F00FF
(1672454739.913492) can0 2D1#030000000000
(1672454739.913852) can0 00D#04E3CF65BF13B842
(1672454739.914567) can0 1F1#AE0E00000800007A
```

Appendix B. Labeled comma-separated values (CSV) samples

B.1. Attack-free

In Listing 23, we provide an example of attack-free traffic—in particular, traffic collected in driving mode—in a labeled `.csv` file. The traffic was collected from the 2011 Chevrolet Traverse.

This is the labeled CSV equivalent of Listing 13 in Section Appendix A.1.

Listing 23 Labeled attack-free traffic (driving mode).

```
1672531371.842441,1E1,0000100000,0
1672531371.843111,0C7,00D4A352,0
1672531371.843342,0F9,031A4002A23FDEFA,0
1672531371.843591,189,4FFF0FFF3000DEFA,0
1672531371.843815,199,4FFF0E70F18F00FF,0
1672531371.8439581,1EB,0141,0
1672531371.844667,1F1,AE0E00000800007A,0
1672531371.845175,0C1,236F7087236A5491,0
1672531371.8453221,1CB,10E600,0
1672531371.845534,0C5,234188A42341E475,0
1672531371.8457692,184,0001000001FF,0
1672531371.845994,1C7,0FFF700003FF3F,0
1672531371.846168,1CD,C7FF07FE7F,0
1672531371.846438,1E5,4600D8EED4FF2500,0
1672531371.8466392,1E9,0FFF000C00260000,0
1672531371.846887,0C9,801D551911000000,0
1672531371.847044,0F1,1C040040,0
1672531371.847256,191,075E075E0761115D,0
1672531371.847515,1ED,41300809067B0800,0
1672531371.847687,1EF,00000920,0
1672531371.847968,2C3,0905066906697200,0
1672531371.848079,2F9,80010F002C,0
1672531371.848239,334,0000,0
1672531371.848416,348,030B0306,0
1672531371.848661,17D,04E000007D000100,0
1672531371.848961,17F,0000000000000000,0
1672531371.849118,34A,03110306,0
1672531371.849538,1F3,0020,0
1672531371.8517342,3C9,0766000000000000,0
1672531371.854999,0C1,33717FE0336B5C56,0
1672531371.855222,0C5,334398013342EC32,0
1672531371.855446,1E5,4600D48E70FF2C00,0
1672531371.8556201,0C7,00DCAFFE,0
1672531371.855855,0F9,031A4002A23FDEFA,0
1672531371.8561,189,8FFF0FFF2FFFDEFA,0
1672531371.8562222,1EB,0124,0
1672531371.8564692,199,8FFF0E70F18E00FF,0
1672531371.856808,0C9,801D4E1C10000000,0
1672531371.857,0F1,28040040,0
1672531371.857222,191,075A075A075E105A,0
1672531371.857444,1ED,413007ED06730800,0
1672531371.857605,1EF,000008BE,0
1672531371.857865,1A1,0000414000001000,0
1672531371.858099,1C3,075A076A00000000,0
1672531371.864171,19D,40003FFF0018105C,0
1672531371.864332,1AF,000000,0
1672531371.8645902,1F5,0202000400000900,0
1672531371.86517,0C1,03728785036D6BE1,0
1672531371.865317,1CB,10D900,0
1672531371.865528,0C5,03449FB80344FBAA,0
```

B.2. Attack

In Listing 24, we provide an example of attack traffic—specifically, fuzzing attack traffic—in a labeled `.csv` file. The traffic was collected from the 2011 Chevrolet Traverse.

This is the labeled CSV equivalent of Listing 17 in Section Appendix A.3.

Listing 24 A labeled fuzzing attack. The red lines indicate attack CAN frames.

```
1672531209.2064872,348,00000000,0
1672531209.2064922,34A,00000000,0
1672531209.2072642,036,F237,1
1672531209.208342,0EE,750F037DB0973413,1
1672531209.208786,0F1,34050040,0
1672531209.2094111,63C,BB98131FFB6BB64D,1
1672531209.209938,19D,C0003FFD000000FF,0
1672531209.209954,1AF,000000,0
1672531209.210492,758,B4E4B84DB744242D,1
1672531209.2110262,1F5,0F0F002100000300,0
1672531209.211039,1EB,0000,0
1672531209.211636,1B9,619E732063822815,1
1672531209.2128332,107,A60ED87316918238,1
1672531209.213259,0C1,0000000000000000,0
1672531209.21398,3AA,9442CD7C6E7A727F,1
1672531209.2143612,0C5,000000000002A384,0
1672531209.214379,184,0002000001FE,0
1672531209.214384,0C9,0000000A00000000,0
1672531209.214388,191,061408F508F50000,0
1672531209.2143931,1C7,0FFFAFFF03FF3F,0
1672531209.2150562,255,BA661E,1
1672531209.215536,1CD,07FF08017F,0
1672531209.215552,1E5,46FFF5A000000A00,0
1672531209.215557,1E9,0000000C00000000,0
1672531209.2155612,0C7,03FE0000,0
1672531209.215565,1ED,0000000000000800,0
1672531209.2162142,665,E9ECB608535FB713,1
1672531209.2167132,0F9,00004000000000FF,0
1672531209.216731,1EF,00000000,0
1672531209.216738,189,CFFF0FFF2FFE00FF,0
1672531209.216743,334,0000,0
1672531209.2167492,199,CFFF0E70F18D00FF,0
1672531209.217368,218,03F7EB2654A8732F,1
1672531209.217899,0F1,00050040,0
1672531209.217909,1F3,8080,0
1672531209.218438,5A8,4F142A7D00B5B738,1
1672531209.2195811,5DA,B8F9DB65,1
1672531209.220752,291,F6F9003C98C3F40E,1
1672531209.2218962,6E7,AE54774792E2CD68,1
1672531209.2223182,1CB,100000,0
1672531209.2229671,335,005D,1
1672531209.2235172,0C1,1000000010000000,0
1672531209.223552,0C5,100000001002A384,0
1672531209.223567,1E5,46FFF5C000000900,0
1672531209.2241352,3A5,17E83D560AA2C903,1
1672531209.224708,1EB,0000,0
1672531209.224719,0C9,0000000D00000000,0
1672531209.224722,191,061408F508F50000,0
1672531209.224724,1ED,0000000000000800,0
1672531209.224726,1EF,00000000,0
```

Appendix C. Detailed results

Here, we provide the detailed results of our 18 traditional machine learning models. We conducted multiple experiments with different sets of parameters; we include only the parameters that resulted in the best performance (in terms of maximum F1-score). The results are tabulated by model and organized by learning style (i.e., traditional machine learning, deep learning, supervised learning, unsupervised learning).

Additionally, the `can-benchmark` repository contains the artifacts generated over the course of our benchmark evaluation. We also provide a Microsoft Excel spreadsheet, `benchmark.xlsx`, in which we meticulously organize the results of our experiments. The `can-benchmark` repository is available here: https://bitbucket.org/brooke-lampe/can-benchmark/src/master/.

Table C.16
Gaussian naive bayes.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.8820	0.9814	0.8820	0.9273	0.8877	0.1123	0.1180	23214	5006275	633115	40066	2071507728	617191930
#1	#2	0.8652	0.9479	0.8652	0.9044	0.8868	0.1132	0.1348	6227	5572327	711423	157940	2071507728	674632590
#1	#3	0.8839	0.9957	0.8839	0.9361	0.8854	0.1146	0.1161	5428	7626878	987572	14927	2071507728	1069338929
#1	#4	0.8906	0.9979	0.8906	0.9411	0.8914	0.1086	0.1094	919	11772352	1433959	12880	2071507728	1671649398
#2	#1	0.9901	0.9982	0.9901	0.9940	0.9908	0.0092	0.0099	3916	13085042	121269	9883	2995791737	1402378958
#2	#2	0.9312	0.9958	0.9312	0.9622	0.9328	0.0672	0.0688	3891	7859552	565826	15295	2995791737	917054299
#2	#3	0.9878	0.9952	0.9878	0.9915	0.9902	0.0098	0.0122	0	12002138	119127	28791	2995791737	1278145039
#2	#4	0.9080	0.9921	0.9080	0.9482	0.9114	0.0886	0.0920	0	4349290	422606	18008	2995791737	565499280
#3	#1	0.9805	0.9632	0.9805	0.9718	0.9991	0.0009	0.0195	2	8405583	7703	159333	2053250668	1018277459
#3	#2	0.9775	0.9559	0.9775	0.9664	1.0000	0.0000	0.0225	2	6697885	160	153916	2053250668	708593149
#3	#3	0.9969	0.9962	0.9969	0.9965	0.9988	0.0012	0.0031	0	9418914	10885	18110	2053250668	977993309
#3	#4	0.9784	0.9583	0.9784	0.9677	1.0000	0.0000	0.0216	1	6746053	18	149221	2053250668	712603019
#4	#1	0.9755	0.9596	0.9755	0.9668	0.9968	0.0032	0.0245	2379	6724253	21818	146843	1711150808	716855819
#4	#2	0.9961	0.9951	0.9961	0.9955	0.9987	0.0013	0.0039	8586	17329134	21721	46195	1711150808	1935236078
#4	#3	0.9730	0.9596	0.9730	0.9656	0.9944	0.0056	0.0270	6311	6660505	37540	147607	1711150808	712036359
#4	#4	0.9758	0.9701	0.9758	0.9674	0.9987	0.0013	0.0242	26438	7951944	10711	187009	1711150808	849491579

Parameters: N/A.

Table C.17
K-nearest neighbors.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9855	0.3376	0.3141	0.3254	0.9931	0.0069	0.6859	19877	5600381	39009	43403	106205148337753	2698926122173
#1	#2	0.9714	0.0009	0.0001	0.0002	0.9967	0.0033	0.9999	19	6263171	20579	164148	106205148337753	6628601627607
#1	#3	0.9976	0.4762	0.0005	0.0010	1.0000	0.0000	0.9995	10	8614439	11	20345	106205148337753	3326301454816
#1	#4	0.9982	0.0190	0.0148	0.0166	0.9992	0.0008	0.9852	204	13195779	10532	13595	106205148337753	11972712642811
#2	#1	0.9988	0.4198	0.4774	0.4467	0.9993	0.0007	0.5226	6587	13197207	9104	7212	218093481844902	4188777687199
#2	#2	0.7930	0.0065	0.5915	0.0128	0.7935	0.2065	0.4085	11349	6685546	1739832	7837	218093481844902	6600135951392
#2	#3	0.9261	0.0000	0.0007	0.0000	0.9283	0.0717	0.9993	19	11252625	868640	28772	218093481844902	4650007219676
#2	#4	0.7658	0.0135	0.8485	0.0265	0.7655	0.2345	0.1515	15279	3652811	1119085	2729	218093481844902	3205390298382
#3	#1	0.9912	0.9983	0.5274	0.6902	1.0000	0.0000	0.4726	84040	8413145	141	75295	201836070397187	2657123832531
#3	#2	0.9461	0.1444	0.2838	0.1914	0.9613	0.0387	0.7162	43680	6439144	258901	110238	201836070397187	6138068209393
#3	#3	0.9980	0.0675	0.0015	0.0030	1.0000	0.0000	0.9985	28	9429412	387	18082	201836070397187	3579896219792
#3	#4	0.9489	0.0284	0.0409	0.0335	0.9690	0.0310	0.9591	6100	6537051	209020	143122	201836070397187	5583217024190
#4	#1	0.9825	0.9638	0.1975	0.3279	0.9998	0.0002	0.8025	29476	6744965	1106	119746	68650714770712	1751927202489
#4	#2	0.7357	0.0059	0.4927	0.0116	0.7365	0.2635	0.5073	26992	12778983	4571872	27789	68650714770712	22966479875097
#4	#3	0.9820	0.9280	0.2177	0.3527	0.9996	0.0004	0.7823	33515	6695444	2601	120403	68650714770712	2006599456078
#4	#4	0.7117	0.0385	0.4191	0.0705	0.7195	0.2805	0.5809	89462	5729122	2233533	123985	68650714770712	11478407659207

Parameters: n_neighbors = 3.

Table C.18
Linear regression.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.3540	0.9680	0.3540	0.5042	0.3467	0.6533	0.6460	117735	2916636	5496650	41600	3544294407	141826800
#1	#2	0.3112	0.9668	0.3112	0.4499	0.2989	0.7011	0.6888	130470	2002190	4695855	23448	3544294407	127655570
#1	#3	0.3467	0.9939	0.3467	0.5136	0.3469	0.6531	0.6533	4432	3271141	6158658	13678	3544294407	159399960
#1	#4	0.3341	0.9758	0.3341	0.4757	0.3204	0.6796	0.6659	142195	2161436	4584635	7027	3544294407	137564940
#2	#1	0.3009	0.9447	0.3009	0.4440	0.2961	0.7039	0.6991	77230	1997231	4748840	71992	2848875729	113877510
#2	#2	0.2208	0.9963	0.2208	0.3575	0.2185	0.7815	0.7792	52599	3790832	13560023	2182	2848875729	304505760
#2	#3	0.2656	0.9423	0.2656	0.3993	0.2585	0.7415	0.7344	88413	1731553	4966492	65505	2848875729	96606370
#2	#4	0.2560	0.9558	0.2560	0.3780	0.2408	0.7592	0.7440	175565	1917644	6045011	37882	2848875729	132705520
#3	#1	0.2096	0.9891	0.2096	0.3309	0.2007	0.7993	0.7904	63280	1131729	4507661	0	2870171858	196197980
#3	#2	0.1668	0.9753	0.1668	0.2483	0.1450	0.8550	0.8332	164167	911368	5372382	0	2870171858	231412030
#3	#3	0.3392	0.9938	0.3392	0.5046	0.3390	0.6610	0.6608	9152	2919967	5694483	11203	2870171858	143956480
#3	#4	0.2466	0.9962	0.2466	0.3949	0.2465	0.7535	0.7534	4717	3255868	9950443	9082	2870171858	222731878
#4	#1	0.3243	0.9985	0.3243	0.4886	0.3238	0.6762	0.6757	11924	4275841	8930470	1875	4962548165	255239910
#4	#2	0.4802	0.9966	0.4802	0.6465	0.4795	0.5205	0.5198	14688	4040135	4385243	4498	4962548165	132193889
#4	#3	0.6591	0.9941	0.6591	0.7926	0.6606	0.3394	0.3409	0	8007795	4113470	28791	4962548165	184774520
#4	#4	0.4382	0.9882	0.4382	0.607	0.4397	0.5603	0.5618	952	2098157	2673739	17056	4962548165	89234350

Parameters: threshold = 0.0.

C.1. Traditional machine learning, supervised

We provide the results of our supervised traditional machine learning models in Tables C.16, C.17, C.18, C.19, C.20, C.21, C.22, C.23, C.24, C.25, C.26, and C.27.

Recall that the structure of each sub-dataset—`set_01`, `set_02`, `set_03`, and `set_04`—is as follows:

1. **train_01:** Train the model
2. **test_01_known_vehicle_known_attack:** Test the model against a known vehicle (seen in training) and known attacks (seen in training)
3. **test_02_unknown_vehicle_known_attack:** Test the model against an unknown vehicle (not seen in training) and known attacks (seen in training)

Table C.19
Logistic regression.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9890	0.9871	0.9890	0.9878	0.9964	0.0036	0.0110	20723	5619213	20177	42557	112252658417	380374780
#1	#2	0.9565	0.9514	0.9565	0.9539	0.9805	0.0195	0.0435	6195	6161349	122401	157972	112252658417	199207470
#1	#3	0.9976	0.9954	0.9976	0.9964	0.9999	0.0001	0.0024	15	8613985	465	20340	112252658417	696710769
#1	#4	0.9976	0.9979	0.9976	0.9977	0.9986	0.0014	0.0024	5	13187768	18543	13794	112252658417	984385909
#2	#1	0.9983	0.9982	0.9983	0.9983	0.9993	0.0007	0.0017	1598	13196556	9755	12201	54372904797	342538899
#2	#2	0.9953	0.9957	0.9953	0.9955	0.9974	0.0026	0.0047	964	8403867	21511	18222	54372904797	293114960
#2	#3	0.9966	0.9953	0.9966	0.9959	0.9990	0.0010	0.0034	0	12108588	12677	28791	54372904797	369723330
#2	#4	0.9922	0.9925	0.9922	0.9924	0.9960	0.0040	0.0078	0	4752757	19139	18008	54372904797	161513620
#3	#1	0.9860	0.9851	0.9860	0.9823	0.9995	0.0005	0.0140	43834	8409073	4213	115501	90624628759	270630190
#3	#2	0.9676	0.9636	0.9676	0.9655	0.9861	0.0139	0.0324	25041	6605205	92840	128877	90624628759	209136650
#3	#3	0.9977	0.9962	0.9977	0.9970	0.9997	0.0003	0.0023	4	9426611	3188	18106	90624628759	318505190
#3	#4	0.9679	0.9570	0.9679	0.9624	0.9893	0.0107	0.0321	29	6673937	72134	149193	90624628759	201148210
#4	#1	0.9784	0.9788	0.9784	0.9677	1.0000	0.0000	0.0216	44	6746071	0	149178	35900328378	199674490
#4	#2	0.9968	0.9947	0.9968	0.9953	1.0000	0.0000	0.0032	232	17350305	550	54549	35900328378	495121370
#4	#3	0.9775	0.9780	0.9775	0.9664	1.0000	0.0000	0.0225	36	6698045	0	153882	35900328378	202264510
#4	#4	0.9739	0.9605	0.9739	0.9611	0.9999	0.0001	0.0261	562	7961996	659	212885	35900328378	248040920

Parameters: N/A.

Table C.20
Linear support vector machine.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9841	0.3018	0.3275	0.3141	0.9915	0.0085	0.6725	20723	5591450	47940	42557	35314797566	344383409
#1	#2	0.9593	0.0538	0.0361	0.0432	0.9834	0.0166	0.9639	5923	6179525	104225	158244	35314797566	354491014
#1	#3	0.9975	0.0131	0.0009	0.0017	0.9998	0.0002	0.9991	19	8613017	1433	20336	35314797566	673546732
#1	#4	0.9942	0.0002	0.0009	0.0003	0.9952	0.0048	0.9991	12	13143123	63188	13787	35314797566	661366602
#2	#1	0.9990	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0	13206311	0	13799	57758151570	664285259
#2	#2	0.9977	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0	8425378	0	19186	57758151570	486901480
#2	#3	0.9976	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0	12121265	0	28791	57758151570	621317192
#2	#4	0.9962	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0	4771896	0	18008	57758151570	291455207
#3	#1	0.9858	0.8829	0.2752	0.4196	0.9993	0.0007	0.7248	43845	8407471	5815	115490	43113677937	492298758
#3	#2	0.9498	0.1412	0.2433	0.1787	0.9660	0.0340	0.7567	37447	6470267	227778	116471	43113677937	413217868
#3	#3	0.9975	0.0009	0.0003	0.0004	0.9994	0.0006	0.9997	5	9424016	5783	18105	43113677937	523456157
#3	#4	0.9436	0.0001	0.0002	0.0002	0.9645	0.0355	0.9998	34	6506490	239581	149188	43113677937	422478071
#4	#1	0.9784	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0	6746071	0	149222	21097322961	415427420
#4	#2	0.9969	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0	17350855	0	54781	21097322961	820916351
#4	#3	0.9775	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0	6698045	0	153918	21097322961	417188365
#4	#4	0.9739	0.0000	0.0000	0.0000	1.0000	0.0000	1.0000	0	7962655	0	213447	21097322961	480077009

Parameters: loss = hinge.

Table C.21
One-class support vector machine.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	-	-	-	-	-	-	-	-	-	-	-	-	-
#1	#2	-	-	-	-	-	-	-	-	-	-	-	-	-
#1	#3	-	-	-	-	-	-	-	-	-	-	-	-	-
#1	#4	-	-	-	-	-	-	-	-	-	-	-	-	-
#2	#1	-	-	-	-	-	-	-	-	-	-	-	-	-
#2	#2	-	-	-	-	-	-	-	-	-	-	-	-	-
#2	#3	-	-	-	-	-	-	-	-	-	-	-	-	-
#2	#4	-	-	-	-	-	-	-	-	-	-	-	-	-
#3	#1	-	-	-	-	-	-	-	-	-	-	-	-	-
#3	#2	-	-	-	-	-	-	-	-	-	-	-	-	-
#3	#3	-	-	-	-	-	-	-	-	-	-	-	-	-
#3	#4	-	-	-	-	-	-	-	-	-	-	-	-	-
#4	#1	-	-	-	-	-	-	-	-	-	-	-	-	-
#4	#2	-	-	-	-	-	-	-	-	-	-	-	-	-
#4	#3	-	-	-	-	-	-	-	-	-	-	-	-	-
#4	#4	-	-	-	-	-	-	-	-	-	-	-	-	-

Parameters: kernel = rbf.
The one-class support vector machine exceeded our 6-day (518400000000000 ns) time limit.

4. **test_03_known_vehicle_unknown_attack:** Test the model against a known vehicle (seen in training) and unknown attacks (not seen in training)
5. **test_04_unknown_vehicle_unknown_attack:** Test the model against an unknown vehicle (not seen in training) and unknown attacks (not seen in training)

Table C.22
Support vector machine.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9720	0.1505	0.3275	0.2062	0.9793	0.0207	0.6725	20723	5522435	116955	42557	116557173711567	20169585048115
#1	#2	0.9618	0.0630	0.0361	0.0459	0.9860	0.0140	0.9639	5923	6195589	88161	158244	116557173711567	22794629836354
#1	#3	0.9976	0.0260	0.0009	0.0018	0.9999	0.0001	0.9991	19	8613737	713	20336	116557173711567	30536179241509
#1	#4	0.9981	0.0007	0.0006	0.0007	0.9992	0.0008	0.9994	8	13195569	10742	13791	116557173711567	46633303631277
#2[a]	#1	-	-	-	-	-	-	-	-	-	-	-	-	-
#2	#2	-	-	-	-	-	-	-	-	-	-	-	-	-
#2	#3	-	-	-	-	-	-	-	-	-	-	-	-	-
#2	#4	-	-	-	-	-	-	-	-	-	-	-	-	-
#3	#1	0.9896	0.9994	0.4429	0.6138	1.0000	0.0000	0.5571	70565	8413244	42	88770	91958405013243	11563361239836
#3	#2	0.9420	0.1549	0.3547	0.2156	0.9555	0.0445	0.6453	54597	6400075	297970	99321	91958405013243	9241437143758
#3	#3	0.9981	0.4848	0.0018	0.0035	1.0000	0.0000	0.9982	32	9429765	34	18078	91958405013243	12744773269736
#3	#4	0.9412	0.0228	0.0410	0.0293	0.9611	0.0389	0.9590	6112	6483827	262244	143110	91958405013243	9293928850140
#4	#1	0.9810	0.9913	0.1254	0.2226	1.0000	0.0000	0.8746	18707	6745907	164	130515	89048060616553	7497330386451
#4	#2	0.8493	0.0103	0.4950	0.0203	0.8504	0.1496	0.5050	27115	14754944	2595911	27666	89048060616553	18922964283852
#4	#3	0.9814	0.9149	0.1872	0.3109	0.9996	0.0004	0.8128	28819	6695365	2680	125099	89048060616553	7447968438532
#4	#4	0.8281	0.0639	0.4089	0.1105	0.8393	0.1607	0.5911	87280	6683287	1279368	126167	89048060616553	8891452202913

Parameters: kernel = rbf.

[a] This experiment exceeded our 6-day (518400000000000 ns) time limit.

Table C.23
Decision tree.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9742	0.0724	0.1123	0.0880	0.9838	0.0162	0.8877	7109	5548288	91102	56171	8271963209	401618047
#1	#2	0.9738	0.0000	0.0000	0.0000	0.9993	0.0007	1.0000	0	6279265	4485	164167	8271963209	471393563
#1	#3	0.9821	0.0022	0.0148	0.0039	0.9844	0.0156	0.9852	301	8479721	134729	20054	8271963209	602445631
#1	#4	0.9827	0.0018	0.0276	0.0033	0.9837	0.0163	0.9724	381	12990683	215628	13418	8271963209	997735598
#2	#1	0.9982	0.3166	0.6010	0.4148	0.9986	0.0014	0.3990	8293	13188413	17898	5506	43693779718	1547987161
#2	#2	0.6617	0.0023	0.3430	0.0046	0.6624	0.3376	0.6570	6580	5580937	2844441	12606	43693779718	751740232
#2	#3	0.6618	0.0069	0.9972	0.0138	0.6610	0.3390	0.0028	28711	8012677	4108588	80	43693779718	1217496104
#2	#4	0.5791	0.0034	0.3757	0.0067	0.5799	0.4201	0.6243	6765	2767142	2004754	11243	43693779718	451008467
#3	#1	0.9963	0.9779	0.8213	0.8928	0.9996	0.0004	0.1787	130857	8410335	2951	28478	29338298907	1040280485
#3	#2	0.7406	0.0424	0.4890	0.0781	0.7464	0.2536	0.5110	75264	4999611	1698434	78654	29338298907	737568595
#3	#3	0.9317	0.0007	0.0257	0.0014	0.9334	0.0666	0.9743	466	8801680	628119	17644	29338298907	1131791308
#3	#4	0.7310	0.0050	0.0582	0.0093	0.7459	0.2541	0.9418	8692	5031918	1714153	140530	29338298907	810102792
#4	#1	0.9880	0.9705	0.4586	0.6229	0.9997	0.0003	0.5414	68433	6743991	2080	80789	23308577039	687957287
#4	#2	0.6006	0.0034	0.4347	0.0068	0.6012	0.3988	0.5653	23813	10430578	6920277	30968	23308577039	1844496718
#4	#3	0.9822	0.8470	0.2510	0.3873	0.9990	0.0010	0.7490	38635	6691065	6980	115283	23308577039	694299037
#4	#4	0.5737	0.0296	0.4828	0.0558	0.5761	0.4239	0.5172	103051	4587474	3375181	110396	23308577039	907885222

Parameters: criterion = entropy, splitter = random, min_samples_split = 2.

Table C.24
Extra trees.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9915	0.8199	0.3046	0.4442	0.9992	0.0008	0.6954	19275	5635157	4233	44005	209531429363	20667658310
#1	#2	0.9740	0.0000	0.0000	0.0000	0.9994	0.0006	1.0000	0	6280034	3716	164167	209531429363	24155191698
#1	#3	0.9976	1.0000	0.0001	0.0003	1.0000	0.0000	0.9999	3	8614450	0	20352	209531429363	31514357019
#1	#4	0.9988	0.0000	0.0000	0.0000	0.9998	0.0002	1.0000	0	13203625	2686	13799	209531429363	49439825750
#2	#1	0.9990	0.5307	0.5210	0.5258	0.9995	0.0005	0.4790	7189	13199954	6357	6610	1048485087501	94353095263
#2	#2	0.5689	0.0039	0.7458	0.0078	0.5685	0.4315	0.2542	14308	4789796	3635582	4878	1048485087501	44998186725
#2	#3	0.7372	0.0089	0.9983	0.0177	0.7366	0.2634	0.0017	28741	8928388	3192877	50	1048485087501	71005505737
#2	#4	0.5222	0.0072	0.9175	0.0142	0.5207	0.4793	0.0825	16523	2484747	2287149	1485	1048485087501	25379076934
#3	#1	0.9952	0.9986	0.7432	0.8522	1.0000	0.0000	0.2568	118425	8413123	163	40910	579481153121	52518736208
#3	#2	0.8910	0.0815	0.3753	0.1339	0.9028	0.0972	0.6247	57763	6047253	650792	96155	579481153121	36366904399
#3	#3	0.9979	0.0596	0.0073	0.0130	0.9998	0.0002	0.9927	132	9427716	2083	17978	579481153121	57253911330
#3	#4	0.8995	0.0167	0.0630	0.0264	0.9180	0.0820	0.9370	9404	6192831	553240	139818	579481153121	35714877783
#4	#1	0.9837	0.9972	0.2467	0.3956	1.0000	0.0000	0.7533	36817	6745968	103	112405	486225104765	36751033345
#4	#2	0.5957	0.0042	0.5433	0.0084	0.5959	0.4041	0.4567	29765	10339035	7011820	25016	486225104765	96779859363
#4	#3	0.9828	0.9911	0.2347	0.3796	1.0000	0.0000	0.7653	36132	6697720	325	117786	486225104765	37676383305
#4	#4	0.5606	0.0277	0.4640	0.0523	0.5632	0.4368	0.5360	99037	4484403	3478252	114410	486225104765	46708815695

Parameters: n_estimators = 50, criterion = gini, min_samples_split = 2.

C.2. Traditional machine learning, unsupervised

We provide the results of our unsupervised traditional machine learning models in Tables C.28, C.29, C.30, and C.31.

C.3. Deep learning, supervised

We provide the results of our supervised deep model—the multi-layer perceptron—in Table C.32.

Table C.25
Gradient boosting.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.6254	0.9892	0.6254	0.7585	0.6212	0.3788	0.3746	63280	3503305	2136085	0	2451403662347	8489738261
#1	#2	0.6626	0.9763	0.6626	0.7739	0.6538	0.3462	0.3374	164079	4108265	2175485	88	2451403662347	8834463042
#1	#3	0.9977	0.9977	0.9977	0.9966	1.0000	0.0000	0.0023	469	8614449	1	19886	2451403662347	11825961968
#1	#4	0.9984	0.9980	0.9984	0.9982	0.9994	0.0006	0.0016	644	13198117	8194	13155	2451403662347	18095133741
#2	#1	0.9992	0.9991	0.9992	0.9991	0.9998	0.0002	0.0008	5942	13203502	2809	7857	4148960101197	18509739315
#2	#2	0.6896	0.9960	0.6896	0.8142	0.6900	0.3100	0.3104	9133	5813919	2611459	10053	4148960101197	10610766902
#2	#3	0.7431	0.9976	0.7431	0.8503	0.7425	0.2575	0.2569	28741	9000552	3120713	50	4148960101197	17166843827
#2	#4	0.6893	0.9952	0.6893	0.8125	0.6889	0.3111	0.3107	14399	3287480	1484416	3609	4148960101197	6031742978
#3	#1	0.9959	0.9959	0.9959	0.9956	1.0000	0.0000	0.0041	124061	8413256	30	35274	2904415225735	9633979740
#3	#2	0.6875	0.9578	0.6875	0.7958	0.6951	0.3049	0.3125	54831	4655559	2042486	99087	2904415225735	7521772601
#3	#3	0.8441	0.9959	0.8441	0.9137	0.8457	0.1543	0.1559	414	7974752	1455047	17696	2904415225735	10332523024
#3	#4	0.6789	0.9514	0.6789	0.7910	0.6915	0.3085	0.3211	16328	4664596	2081475	132894	2904415225735	7476481741
#4	#1	0.9800	0.9771	0.9800	0.9722	0.9996	0.0004	0.0200	14247	6743146	2925	134975	2246450290838	7599878712
#4	#2	0.8378	0.9945	0.8378	0.9088	0.8393	0.1607	0.1622	20346	14562744	2788111	34435	2246450290838	22651220622
#4	#3	0.9804	0.9760	0.9804	0.9756	0.9975	0.0025	0.0196	36158	6681301	16744	117760	2246450290838	7745712710
#4	#4	0.8217	0.9534	0.8217	0.8798	0.8359	0.1641	0.1783	61914	6655982	1306673	151533	2246450290838	10831451554

Parameters: loss = deviance, learning_rate = 0.1, n_estimators = 100, criterion = friedman_mse, min_samples_split = 2.

Table C.26
Isolation forest.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9846	0.9779	0.9846	0.9812	0.9956	0.0044	0.0154	0	5614590	24800	63280	197244480367	88099567128
#1	#2	0.9550	0.9492	0.9550	0.9521	0.9799	0.0201	0.0450	0	6157449	126301	164167	197244480367	100337823607
#1	#3	0.9969	0.9957	0.9969	0.9963	0.9992	0.0008	0.0031	928	8607461	6989	19427	197244480367	134518109020
#1	#4	0.9881	0.9980	0.9881	0.9930	0.9890	0.0110	0.0119	1131	13061670	144641	12668	197244480367	202233230770
#2	#1	0.9990	0.9990	0.9990	0.9986	1.0000	0.0000	0.0010	695	13206294	17	13104	324118674127	206854622230
#2	#2	0.9900	0.9956	0.9900	0.9928	0.9921	0.0079	0.0100	1156	8358875	66503	18030	324118674127	119965866722
#2	#3	0.9976	0.9953	0.9976	0.9964	1.0000	0.0000	0.0024	0	12121229	36	28791	324118674127	185499306577
#2	#4	0.9874	0.9925	0.9874	0.9899	0.9911	0.0089	0.0126	0	4729471	42425	18008	324118674127	68289354860
#3	#1	0.9809	0.9672	0.9809	0.9723	0.9993	0.0007	0.0191	1597	8407181	6105	157738	226093669303	134993492743
#3	#2	0.9756	0.9574	0.9756	0.9658	0.9979	0.0021	0.0244	1139	6683950	14095	152779	226093669303	100789302007
#3	#3	0.9973	0.9962	0.9973	0.9967	0.9992	0.0008	0.0027	0	9422289	7510	18110	226093669303	146008065708
#3	#4	0.9761	0.9576	0.9761	0.9666	0.9977	0.0023	0.0239	338	6730333	15738	148884	226093669303	99429036200
#4	#1	0.9774	0.9578	0.9774	0.9672	0.9990	0.0010	0.0226	177	6739416	6655	149045	171923906654	103616521318
#4	#2	0.9840	0.9937	0.9840	0.9888	0.9870	0.0130	0.0160	941	17125712	225143	53840	171923906654	260236126394
#4	#3	0.9764	0.9568	0.9764	0.9660	0.9988	0.0012	0.0236	445	6689938	8107	153473	171923906654	102870051455
#4	#4	0.9628	0.9490	0.9628	0.9558	0.9884	0.0116	0.0372	2074	7870193	92462	211373	171923906654	120766222169

Parameters: n_estimators = 50, contamination = 0.001.

Table C.27
Random forest.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9879	0.0088	0.0008	0.0015	0.9990	0.0010	0.9992	52	5633511	5879	63228	763511969876	38249637250
#1	#2	0.9720	0.0000	0.0000	0.0000	0.9974	0.0026	1.0000	0	6267567	16183	164167	763511969876	42700298556
#1	#3	0.9976	1.0000	0.0004	0.0009	1.0000	0.0000	0.9996	9	8614450	0	20346	763511969876	56868215083
#1	#4	0.9988	0.0011	0.0001	0.0003	0.9999	0.0001	0.9999	2	13204519	1792	13797	763511969876	88901090110
#2	#1	0.9991	0.6009	0.5464	0.5724	0.9996	0.0004	0.4536	7540	13201303	5008	6259	3114043843818	136611348151
#2	#2	0.7176	0.0032	0.3972	0.0064	0.7183	0.2817	0.6028	7620	6052223	2373155	11566	3114043843818	72486334580
#2	#3	0.7984	0.0116	0.9983	0.0229	0.7979	0.2021	0.0017	28741	9671773	2449492	50	3114043843818	112412945759
#2	#4	0.6682	0.0094	0.8329	0.0185	0.6675	0.3325	0.1671	14998	3185400	1586496	3010	3114043843818	42092086883
#3	#1	0.9963	0.9998	0.8020	0.8900	1.0000	0.0000	0.1980	127785	8413257	29	31550	2020852818255	82556777120
#3	#2	0.8186	0.0449	0.3489	0.0795	0.8294	0.1706	0.6511	53705	5555351	1142694	100213	2020852818255	58518832066
#3	#3	0.9945	0.0109	0.0210	0.0143	0.9963	0.0037	0.9790	380	9395302	34497	17730	2020852818255	90810701195
#3	#4	0.8150	0.0063	0.0484	0.0112	0.8319	0.1681	0.9516	7226	5612130	1133941	141996	2020852818255	58375075253
#4	#1	0.9848	0.9977	0.2961	0.4566	1.0000	0.0000	0.7039	44181	6745969	102	105041	1658659682691	59969033775
#4	#2	0.8023	0.0024	0.1502	0.0048	0.8044	0.1956	0.8498	8227	13956460	3394395	46554	1658659682691	159904765108
#4	#3	0.9839	0.9973	0.2822	0.4399	1.0000	0.0000	0.7178	43437	6697929	116	110481	1658659682691	60600924460
#4	#4	0.7762	0.0366	0.2991	0.0652	0.7890	0.2110	0.7009	63838	6282483	1680172	149609	1658659682691	75674656782

Parameters: n_estimators = 100, criterion = gini, min_samples_split = 2.

Table C.28
K-means clustering.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.8564	0.9763	0.8564	0.9124	0.8660	0.1340	0.1436	0	4883881	755509	63280	97416359356	638434302
#1	#2	0.8654	0.9467	0.8654	0.9042	0.8880	0.1120	0.1346	0	5580250	703500	164167	97416359356	709549141
#1	#3	0.8658	0.9970	0.8658	0.9258	0.8660	0.1340	0.1342	15172	7460471	1153979	5183	97416359356	963140938
#1	#4	0.8872	0.9985	0.8872	0.9392	0.8875	0.1125	0.1128	8883	11720655	1485656	4916	97416359356	1463804724
#2	#1	0.8872	0.9985	0.8872	0.9392	0.8875	0.1125	0.1128	8811	11720655	1485656	4988	130622830521	1447033632
#2	#2	0.7979	0.9961	0.7979	0.8855	0.7987	0.2013	0.2021	8061	6729582	1695796	11125	130622830521	931602718
#2	#3	0.8854	0.9950	0.8854	0.9370	0.8875	0.1125	0.1146	0	10757395	1363870	28791	130622830521	1351110187
#2	#4	0.7955	0.9916	0.7955	0.8828	0.7985	0.2015	0.2045	0	3810356	961540	18008	130622830521	536446070
#3	#1	0.1243	0.9459	0.1243	0.1975	0.1120	0.8880	0.8757	123697	942015	7471271	35638	108539371536	942543350
#3	#2	0.2282	0.9630	0.2282	0.3445	0.2138	0.7862	0.7718	131445	1432045	5266000	22473	108539371536	748779763
#3	#3	0.1123	0.9855	0.1123	0.2008	0.1120	0.8880	0.8877	4614	1055914	8373885	13496	108539371536	1039551987
#3	#4	0.2297	0.9741	0.2297	0.3453	0.2137	0.7863	0.7703	142038	1441653	5304418	7184	108539371536	764330221
#4	#1	0.7698	0.9530	0.7698	0.8511	0.7857	0.2143	0.2302	7344	5300665	1445406	141878	49276596864	757799028
#4	#2	0.8861	0.9950	0.8861	0.9367	0.8875	0.1125	0.1139	25670	15398301	1952554	29111	49276596864	1914475119
#4	#3	0.7708	0.9541	0.7708	0.8512	0.7851	0.2149	0.2292	22922	5258890	1439155	130996	49276596864	757572379
#4	#4	0.8762	0.9606	0.8762	0.9130	0.8875	0.1125	0.1238	96903	7066664	895991	116544	49276596864	901783586

Parameters: n_clusters = 2, algorithm = full.

Table C.29
Mini-batch K-means clustering.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.4681	0.9669	0.4681	0.6304	0.4729	0.5271	0.5319	2491	2666944	2972446	60789	7678285764	3796294688
#1	#2	0.4969	0.9271	0.4969	0.6470	0.5099	0.4901	0.5031	88	3204152	3079598	164079	7678285764	4126732218
#1	#3	0.4741	0.9969	0.4741	0.6408	0.4732	0.5268	0.5259	17101	4076409	4538041	3254	7678285764	5800975603
#1	#4	0.4748	0.9984	0.4748	0.6429	0.4745	0.5255	0.5252	10415	6267005	6939306	3384	7678285764	9274790788
#2	#1	0.8872	0.9985	0.8872	0.9392	0.8875	0.1125	0.1128	8808	11720655	1485656	4991	13176242457	8605857753
#2	#2	0.8103	0.9961	0.8103	0.8931	0.8112	0.1888	0.1897	8049	6834386	1590992	11137	13176242457	5676364082
#2	#3	0.8854	0.9950	0.8854	0.9370	0.8875	0.1125	0.1146	0	10757395	1363870	28791	13176242457	8517801026
#2	#4	0.8085	0.9916	0.8085	0.8907	0.8115	0.1885	0.1915	0	3872458	899438	18008	13176242457	3216584183
#3	#1	0.8757	0.9661	0.8757	0.9172	0.8880	0.1120	0.1243	35659	7471271	942015	123676	8493144777	5750632132
#3	#2	0.7712	0.9541	0.7712	0.8514	0.7856	0.2144	0.2288	22492	5261861	1436184	131426	8493144777	4609698464
#3	#3	0.8877	0.9976	0.8877	0.9387	0.8880	0.1120	0.1123	13496	8373885	1055914	4614	8493144777	6304545321
#3	#4	0.7698	0.9529	0.7698	0.8511	0.7857	0.2143	0.2302	7190	5300665	1445406	142032	8493144777	4543563638
#4	#1	0.7597	0.9526	0.7597	0.8448	0.7754	0.2246	0.2403	7399	5230985	1515086	141823	7006960680	4772947871
#4	#2	0.8862	0.9950	0.8862	0.9367	0.8875	0.1125	0.1138	25836	15398301	1952554	28945	7006960680	13150643666
#4	#3	0.7608	0.9538	0.7608	0.8449	0.7748	0.2252	0.2392	23086	5189956	1508089	130832	7006960680	4683530105
#4	#4	0.8762	0.9607	0.8762	0.9130	0.8875	0.1125	0.1238	97496	7066664	895991	115951	7006960680	5598694588

Parameters: n_clusters = 2, batch_size = 512.

Table C.30
BIRCH.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9889	0.9779	0.9889	0.9834	1.0000	0.0000	0.0111	0	5639390	0	63280	214197454658	605665184
#1	#2	0.9745	0.9497	0.9745	0.9620	1.0000	0.0000	0.0255	0	6283750	0	164167	214197454658	662717530
#1	#3	0.9976	0.9953	0.9976	0.9965	1.0000	0.0000	0.0024	0	8614450	0	20355	214197454658	1004912418
#1	#4	0.9990	0.9979	0.9990	0.9984	1.0000	0.0000	0.0010	0	13206311	0	13799	214197454658	1171693676
#2	#1	0.9990	0.9979	0.9990	0.9984	1.0000	0.0000	0.0010	0	13206311	0	13799	297512429069	1163219087
#2	#2	0.9977	0.9955	0.9977	0.9966	1.0000	0.0000	0.0023	0	8425378	0	19186	297512429069	814946104
#2	#3	0.9976	0.9953	0.9976	0.9964	1.0000	0.0000	0.0024	0	12121265	0	28791	297512429069	1120011064
#2	#4	0.9962	0.9925	0.9962	0.9944	1.0000	0.0000	0.0038	0	4771896	0	18008	297512429069	544038241
#3	#1	0.9814	0.9632	0.9814	0.9722	1.0000	0.0000	0.0186	0	8413286	0	159335	283565479448	834557803
#3	#2	0.9775	0.9556	0.9775	0.9664	1.0000	0.0000	0.0225	0	6698045	0	153918	283565479448	681784890
#3	#3	0.9981	0.9962	0.9981	0.9971	1.0000	0.0000	0.0019	0	9429799	0	18110	283565479448	862777662
#3	#4	0.9784	0.9572	0.9784	0.9677	1.0000	0.0000	0.0216	0	6746071	0	149222	283565479448	729009648
#4	#1	0.9784	0.9572	0.9784	0.9677	1.0000	0.0000	0.0216	0	6746071	0	149222	161254066683	678632660
#4	#2	0.9969	0.9937	0.9969	0.9953	1.0000	0.0000	0.0031	0	17350855	0	54781	161254066683	1466783064
#4	#3	0.9775	0.9556	0.9775	0.9664	1.0000	0.0000	0.0225	0	6698045	0	153918	161254066683	664297692
#4	#4	0.9739	0.9485	0.9739	0.9610	1.0000	0.0000	0.0261	0	7962655	0	213447	161254066683	812460288

Parameters: threshold = 0.5, branching_factor = 25, n_clusters = 2.

Table C.31
Local outlier factor.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9254	0.9728	0.9254	0.9465	0.9343	0.0657	0.0746	72077	7860912	552374	87258	53788202512612	2439824879120
#1	#2	0.1714	0.9557	0.1714	0.2640	0.1560	0.8440	0.8286	129536	1044762	5653283	24382	53788202512612	7698684672802
#1	#3	0.9386	0.9965	0.9386	0.9665	0.9400	0.0600	0.0614	4080	8863608	566191	14030	53788202512612	2504114371275
#1	#4	0.1661	0.8882	0.1661	0.2726	0.1644	0.8356	0.8339	36488	1108865	5637206	112734	53788202512612	7995192332965
#2	#1	0.9081	0.9559	0.9081	0.9314	0.9280	0.0720	0.0919	1753	6260040	486031	147469	59034611309148	1175546111752
#2	#2	0.4353	0.9927	0.4353	0.6039	0.4353	0.5647	0.5647	23302	7552983	9797872	31479	59034611309148	18412260067057
#2	#3	0.8891	0.9569	0.8891	0.9210	0.9067	0.0933	0.1109	18532	6073430	624615	135386	59034611309148	1427622552072
#2	#4	0.4468	0.9404	0.4468	0.5970	0.4481	0.5519	0.5532	84533	3568176	4394479	128914	59034611309148	7705037949598
#3	#1	0.9086	0.9781	0.9086	0.9417	0.9178	0.0822	0.0914	5627	5175888	463502	57653	29212871323360	1114523090819
#3	#2	0.1548	0.8408	0.1548	0.2602	0.1579	0.8421	0.8452	6347	992043	5291707	157820	29212871323360	4107012193646
#3	#3	0.9194	0.9965	0.9194	0.9557	0.9202	0.0798	0.0806	11101	7927346	687104	9254	29212871323360	1758614710997
#3	#4	0.3703	0.9978	0.3703	0.5395	0.3701	0.6299	0.6297	8057	4887587	8318724	5742	29212871323360	7697970340249
#4	#1	0.9552	0.9981	0.9552	0.9760	0.9560	0.0440	0.0448	2490	12624813	581498	11309	100295280221578	4241296731211
#4	#2	0.5491	0.9942	0.5491	0.7072	0.5500	0.4500	0.4509	2692	4633997	3791381	16494	100295280221578	9670662153028
#4	#3	0.9501	0.9952	0.9501	0.9721	0.9524	0.0476	0.0499	63	11543811	577454	28728	100295280221578	3849191938482
#4	#4	0.5451	0.9896	0.5451	0.7029	0.5471	0.4529	0.4549	585	2610520	2161376	17423	100295280221578	5465604178764

Parameters: n_neighbors = 20, algorithm = kd_tree, contamination = auto.

Table C.32
Multi-layer perceptron.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.9788	0.9837	0.9788	0.9811	0.9861	0.0139	0.0212	20723	5560852	78538	42557	3088090156514	11708139937
#1	#2	0.9666	0.9529	0.9666	0.9594	0.9908	0.0092	0.0334	6227	6226232	57518	157940	3088090156514	13276672102
#1	#3	0.9974	0.9953	0.9974	0.9964	0.9998	0.0002	0.0026	11	8612758	1692	20344	3088090156514	17550210412
#1	#4	0.9973	0.9979	0.9973	0.9976	0.9983	0.0017	0.0027	204	13184381	21930	13595	3088090156514	27715472855
#2	#1	0.9992	0.9990	0.9992	0.9991	0.9997	0.0003	0.0008	6272	13202688	3623	7527	6061045048456	27148087661
#2	#2	0.8957	0.9963	0.8957	0.9428	0.8968	0.1032	0.1043	8519	7555528	869850	10667	6061045048456	17240834965
#2	#3	0.9642	0.9952	0.9642	0.9795	0.9665	0.0335	0.0358	19	11715522	405743	28772	6061045048456	25427346349
#2	#4	0.8623	0.9921	0.8623	0.9226	0.8654	0.1346	0.1377	880	4129448	642448	17128	6061045048456	9809416007
#3	#1	0.9930	0.9930	0.9930	0.9922	1.0000	0.0000	0.0070	99207	8413055	231	60128	2675529467831	17538778400
#3	#2	0.9779	0.9734	0.9779	0.9750	0.9935	0.0065	0.0221	45840	6654547	43498	108078	2675529467831	14086705828
#3	#3	0.9981	0.9963	0.9981	0.9971	1.0000	0.0000	0.0019	14	9429578	221	18096	2675529467831	19263949489
#3	#4	0.9742	0.9601	0.9742	0.9665	0.9951	0.0049	0.0258	4232	6712976	33095	144990	2675529467831	14168803955
#4	#1	0.9799	0.9787	0.9799	0.9715	0.9999	0.0001	0.0201	11713	6745070	1001	137509	2655845001046	13873431844
#4	#2	0.9284	0.9944	0.9284	0.9599	0.9306	0.0694	0.0716	13826	16145941	1204914	40955	2655845001046	36039415208
#4	#3	0.9808	0.9788	0.9808	0.9745	0.9994	0.0006	0.0192	26513	6693989	4056	127405	2655845001046	13922701709
#4	#4	0.9071	0.9553	0.9071	0.9293	0.9247	0.0753	0.0929	53603	7362785	599870	159844	2655845001046	16551117389

Parameters: activation = logistic, solver = adam, alpha = 0.0001, learning_rate = constant.

Table C.33
Restricted Boltzmann machine.

Data-set	Sub-set	Acc.	Pre.	Rec. (TPR)	F1-score	TNR	FPR	FNR	TP	TN	FP	FN	Training Time (ns)	Testing Time (ns)
#1	#1	0.0111	0.0001	0.0111	0.0002	0.0000	1.0000	0.9889	63280	0	5639390	0	916661700320	9736619899
#1	#2	0.0255	0.0006	0.0255	0.0013	0.0000	1.0000	0.9745	164167	0	6283750	0	916661700320	10970949396
#1	#3	0.0024	0.0000	0.0024	0.0000	0.0000	1.0000	0.9976	20355	0	8614450	0	916661700320	14603996262
#1	#4	0.0010	0.0000	0.0010	0.0000	0.0000	1.0000	0.9990	13799	0	13206311	0	916661700320	22041127093
#2	#1	0.0010	0.0000	0.0010	0.0000	0.0000	1.0000	0.9990	13799	0	13206311	0	1354503991934	24294537797
#2	#2	0.0023	0.0000	0.0023	0.0000	0.0000	1.0000	0.9977	19186	0	8425378	0	1354503991934	15421833893
#2	#3	0.0024	0.0000	0.0024	0.0000	0.0000	1.0000	0.9976	28791	0	12121265	0	1354503991934	22534199837
#2	#4	0.0038	0.0000	0.0038	0.0000	0.0000	1.0000	0.9962	18008	0	4771896	0	1354503991934	8979004213
#3	#1	0.0186	0.0003	0.0186	0.0007	0.0000	1.0000	0.9814	159335	0	8413286	0	930702274084	14362685904
#3	#2	0.0225	0.0005	0.0225	0.0010	0.0000	1.0000	0.9775	153918	0	6698045	0	930702274084	11491870551
#3	#3	0.0019	0.0000	0.0019	0.0000	0.0000	1.0000	0.9981	18110	0	9429799	0	930702274084	15990152359
#3	#4	0.0216	0.0005	0.0216	0.0009	0.0000	1.0000	0.9784	149222	0	6746071	0	930702274084	11655323670
#4	#1	0.1171	0.9575	0.1171	0.1781	0.0998	0.9002	0.8829	134239	673389	6072682	14983	761974625735	12918669375
#4	#2	0.1024	0.9937	0.1024	0.1811	0.0999	0.9001	0.8976	49267	1733885	15616970	5514	761974625735	31999954558
#4	#3	0.1181	0.9559	0.1181	0.1786	0.1002	0.8998	0.8819	138369	670951	6027094	15549	761974625735	12839847161
#4	#4	0.1209	0.9492	0.1209	0.1780	0.1000	0.9000	0.8791	192161	796258	7166397	21286	761974625735	15244361051

Parameters: n_components = 16, learning_rate = 0.1, batch_size = 20, threshold = -0.0001.

C.4. Deep learning, unsupervised

We provide the results of our unsupervised deep model—the restricted Boltzmann machine—in Table C.33.

Appendix D. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cose.2024.103777.

References

8devices, 2023. Korlan usb2can. 8devices.

Agency, E.P., 1999. Control of air pollution from new motor vehicles; compliance programs for new light-duty vehicles and light-duty trucks. Natl. Arch.

Bozdal, M., Samie, M., Aslam, S., Jennions, I., 2020. Evaluation of can bus security challenges. Sensors 20.

Brownlee, J., 2020. A gentle introduction to imbalanced classification. Mach. Learn. Mastery.

Buscemi, A., Turcanu, I., Castignani, G., Panchenko, A., Engel, T., Shin, K.G., 2023. A survey on controller area network reverse engineering. IEEE Commun. Surv. Tutor.

Cai, Z., Wang, A., Zhang, W., 2019. 0-days & mitigations: roadways to exploit and secure connected bmw cars. BlackHat.

can-utils Contributors, 2023. can-utils. linux-can.

Community, T.L.K.D., 2023. Socketcan - controller area network. Linux Kernel.

Daily, J., Van, D., 2021. Secure controller area network logging. SAE Technical Paper.

de Menezes Lourenço, M.A., Eckert, J.J., Silva, F.L., Santiciolli, F.M., Silva, L.C.A., 2022. Vehicle and twin-roller chassis dynamometer model considering slip tire interactions. Mech. Based Des. Struct. Mach.

Dupont, G., Lekidis, A., den Hartog, J., Etalle, S., 2019. Automotive Controller Area Network (CAN) Bus Intrusion Dataset v2. 4TU.Centre for Research Data.

Dürrwang, J., Braun, J., Rumez, M., Kriesten, R., Pretschner, A., 2018. Enhancement of automotive penetration testing with threat analyses results. SAE Int. J. Transp. Cybersecur. Priv. 1, 91–112.

Everett, C.E., McCoy, D., 2013. Octane (open car testbed and network experiments): bringing cyber-physical security research to researchers and students. In: 6th Workshop on Cyber Security Experimentation and Test (CSET 13).

Falch, M., 2022. Obd2 explained - a simple intro [2023]. CSS Electron.

Foster, I., Koscher, K., 2015. Exploring controller area networks. ;login: 40, 6–10.

Foster, I., Prudhomme, A., Koscher, K., Savage, S., 2015. Fast and vulnerable: a story of telematic failures. In: Proceedings of the 9th USENIX Conference on Offensive Technologies (WOOT '15).

Guo, H., Liu, H., Wu, C., Zhi, W., Xiao, Y., She, W., 2016. Logistic discrimination based on g-mean and f-measure for imbalanced problem. J. Intell. Fuzzy Syst. 31, 1155–1166.

Han, M.L., Kwak, B.I., Kim, H.K., 2018. Anomaly intrusion detection method for vehicular networks based on survival analysis. Veh. Commun. 14, 52–63.

Hanselmann, M., Strauss, T., Dormann, K., Ulmer, H., 2020. Canet: an unsupervised intrusion detection system for high dimensional can bus data. IEEE Access 8, 58194–58205.

Kaiser, C., Stocker, A., Festl, A., 2019. Automotive can bus data: an example dataset from the aegis big data project. OpenAIRE.

Kang, H., Kwak, B.I., Lee, Y.H., Lee, H., Lee, H., Kim, H.K., 2021. Car hacking and defense competition on in-vehicle network. In: Third International Workshop on Automotive and Autonomous Vehicle Security.

Kang, M.-J., Kang, J.-W., 2016. Intrusion detection system using deep neural network for in-vehicle network security. PLoS ONE.

Karopoulos, G., Kambourakis, G., Chatzoglou, E., Hernández-Ramos, J.L., Kouliaridis, V., 2022. Demystifying in-vehicle intrusion detection systems: a survey of surveys and a meta-taxonomy. Electronics 11, 1072.

Koscher, K., Czeskis, A., Roesner, F., Shwetak Patel, T.K., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., 2010. Experimental security analysis of a modern automobile. In: IEEE Symposium on Security and Privacy.

Kubat, M., Matwin, S., 1997. Addressing the curse of imbalanced training sets: one-sided selection. In: International Conference on Machine Learning.

Lab, C., 2017. Datasets. Veh. Secur. Res.

Lakhal, N.M.B., Nasri, O., Adouane, L., Slama, J.B.H., 2021. Controller area network reliability: overview of design challenges and safety related perspectives of future transportation systems. IET Intell. Transp. Syst. 14, 1727–1739.

Lampe, B., 2023a. can-csv. Bitbucket.

Lampe, B., 2023b. can-dataset. Bitbucket.

Lampe, B., 2023c. can-log. Bitbucket.

Lampe, B., 2023d. can-ml. Bitbucket.

Lampe, B., 2023e. can-train-and-test. Bitbucket.

Lampe, B.E., 2023f. can-csv. DTU Data.

Lampe, B.E., 2023g. can-dataset. DTU Data.

Lampe, B.E., 2023h. can-log. DTU Data.

Lampe, B.E., 2023i. can-ml. DTU Data.

Lampe, B.E., 2023j. can-train-and-test. DTU Data.

Lampe, B., Meng, W., 2022. Ids for can: a practical intrusion detection system for can bus security. In: 2022 IEEE Global Communications Conference (GLOBECOM 2022).

Lampe, B., Meng, W., 2023a. can-logic: automotive intrusion detection via temporal logic. In: 13th International Conference on the Internet of Things (IoT 2023).

Lampe, B., Meng, W., 2023b. Intrusion detection in the automotive domain: a comprehensive review. IEEE Commun. Surv. Tutor. 25, 2356–2426.

Lampe, B., Meng, W., 2023c. A survey of deep learning-based intrusion detection in automotive applications. Expert Syst. Appl. 221, 119771.

Lampe, B., Meng, W., 2023d. can-train-and-test: a new can intrusion detection dataset. In: 2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall).

Lee, H., Jeong, S.H., Kim, H.K., 2017. Otids: a novel intrusion detection system for in-vehicle network by using remote frame. In: 2017 15th Annual Conference on Privacy, Security and Trust (PST).

Marx, S.E., Luck, J.D., Pitla, S.K., Hoy, R.M., 2016. Comparing various hardware/software solutions and conversion methods for controller area network (can) bus data collection. Comput. Electron. Agric. 128, 141–148.

Miller, C., Valasek, C., 2014. Adventures in automotive networks and control units. IOActive.

Miller, C., Valasek, C., 2015a. Remote exploitation of an unaltered passenger vehicle. IOActive.

Miller, C., Valasek, C., 2015b. Remote exploitation of an unaltered passenger vehicle. BlackHat.

Miller, C., Valasek, C., 2016a. Advanced can injection techniques for vehicle networks. BlackHat.

Miller, C., Valasek, C., 2016b. Can message injection. Illmatics.

Nie, S., Liu, L., Du, Y., 2017. Free-fall: hacking tesla from wireless to can bus. BlackHat.

Novotná, A., 2023. What is an electronic control unit (ecu)? AutoPi.

of Turku, U., 2021. Can bus dataset collected from a heavy-duty truck. Fairdata.

pandas Contributors, 2023. pandas. pandas.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, Édouard, 2011. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.

Pollicino, F., Stabili, D., Marchetti, M., 2023. Performance comparison of timing-based anomaly detectors for controller area network: a reproducible study. arXiv.

python-can Contributors, 2023a. python-can. Read Docs.

python-can Contributors, 2023b. python-can 4.2.1. The Python Package Index (PyPI).

Rajapaksha, S., Kalutarage, H., Al-Kadri, M.O., Petrovski, A., Madzudzo, G., Cheah, M., 2023. Ai-based intrusion detection systems for in-vehicle networks: a survey. ACM Comput. Surv. 55, 1–40.

Sami, M., 2019. Intrusion detection in can bus. IEEE Dataport.

Sami, M., Ibarra, M., Esparza, A.C., Al-Jufout, S., Aliasgari, M., Mozumdar, M., 2020. Rapid, multi-vehicle and feed-forward neural network based intrusion detection system for controller area network bus. In: 2020 IEEE Green Energy and Smart Systems Conference (IGESSC).

Seo, E., Song, H.M., Kim, H.K., 2018. Gids: gan based intrusion detection system for in-vehicle network. In: 2018 16th Annual Conference on Privacy, Security and Trust (PST).

Song, H.M., Kim, H.K., 2020. Discovering can specification using on-board diagnostics. IEEE Des. Test 38, 93–103.

Song, H.M., Woo, J., Kim, H.K., 2020. In-vehicle network intrusion detection using deep convolutional neural network. Veh. Commun. 21, 100198.

Stabili, D., Marchetti, M., 2019. Detection of missing can messages through inter-arrival time analysis. In: 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall).

Stabili, D., Ferretti, L., Andreolini, M., Marchetti, M., 2022. Daga: detecting attacks to in-vehicle networks via n-gram analysis. IEEE Trans. Veh. Technol. 71, 11540–11554.

Sun, X., Meng, W., Chiu, W.-Y., Lampe, B., 2022. Tdl-ids: towards a transfer deep learning based intrusion detection system. In: 2022 IEEE Global Communications Conference (GLOBECOM 2022).

Tindell, K., 2023. Can injection: keyless car theft. Canis Automot. Labs.

Vahidi, A., Rosenstatter, T., Mowla, N.I., 2022. Systematic evaluation of automotive intrusion detection datasets. In: Proceedings of the 6th ACM Computer Science in Cars Symposium (CSCS '22).

Verma, M.E., Iannacone, M.D., Bridges, R.A., Hollifield, S.C., Kay, B., Combs, F.L., 2020. Road: the real ornl automotive dynamometer controller area network intrusion detection dataset (with a comprehensive can ids dataset survey & guide). arXiv.

Verma, M.E., Iannacone, M.D., Bridges, R.A., Hollifield, S.C., Moriano, P., Kay, B., Combs, F.L., 2022. Addressing the lack of comparability & testing in can intrusion detection research: a comprehensive guide to can ids data & introduction of the road dataset. IEEE Trans. Veh. Technol.

Wu, W., Li, R., Xie, G., An, J., Bai, Y., Zhou, J., Li, K., 2019. A survey of intrusion detection for in-vehicle networks. IEEE Trans. Intell. Transp. Syst. 21, 919–933.

Zago, M., Longari, S., Tricarico, A., Carminati, M., Pérez, M.G., Pérez, G.M., Zanero, S., 2020a. Recan data - reverse engineering of controller area networks. Mendeley Data.

Zago, M., Longari, S., Tricarico, A., Carminati, M., Pérez, M.G., Pérez, G.M., Zanero, S., 2020b. Recan – dataset for reverse engineering of controller area networks. Data Brief 29, 105149.

Zalman, R., 2017. Chapter 8 - rugged autonomous vehicles. In: Vega, A., Bose, P., Buyuktosunoglu, A. (Eds.), Rugged Embedded Systems. Morgan Kaufmann, Boston, pp. 237–266.

Brooke Lampe is a PhD student at the Technical University of Denmark. She is currently investigating automotive intrusion detection, particularly for connected and autonomous vehicles (CAVs). Her research interests include automotive security, intrusion detection, deep learning, and blockchain.

Weizhi Meng is an Associate Professor in the Department of Applied Mathematics and Computer Science at the Technical University of Denmark (DTU). He obtained his Ph.D. degree in Computer Science from the City University of Hong Kong. Prior to joining DTU, he worked as Research Scientist in Institute for Infocomm Research, A*Star, Singapore. He won the Outstanding Academic Performance Award during his doctoral study, and is a recipient of the Hong Kong Institution of Engineers (HKIE) Outstanding Paper Award for Young Engineers/Researchers in both 2014 and 2017. He received the IEEE

ComSoc Best Young Researcher Award for Europe, Middle East, & Africa Region (EMEA) in 2020, and the IEEE MGA Young Professionals Achievement Award in 2020 for his contributions to leading activities in Denmark and Region 8. His primary research interests are intersections between Artificial Intelligence, Cyber Security and Blockchain technology. He serves as associate editors / editorial board members for many reputed journals such as IEEE Transactions on Dependable and Secure Computing, as well as General Chair / Program Committee Chair for various international conferences such as ACM CCS 2023 and ESORICS 2022. He is directing the SPTAGE Lab at DTU.