

can-sleuth: Sleuthing Out the Capabilities, Limitations, and Performance Impacts of Automotive Intrusion Detection Datasets

Brooke Kidmose · Andreas Kidmose · Weizhi Meng

Received: date / Accepted: date

Abstract Modern automobiles are made up of networks of computers, one of which is the inherently insecure Controller Area Network (CAN). Over the years, automotive security has been enhanced by secure gateways and new protocols such as automotive Ethernet, but the CAN protocol has remained the weak link. Automotive researchers have been exploring intrusion detection systems (IDSs) as a potential solution to the problem of CAN bus *insecurity*. To build and evaluate an IDS, however, researchers need adequate training and testing data.

In this paper, we analyze and compare the following automotive intrusion detection datasets: (1) HCRL Car Hacking, (2) HCRL Survival Analysis, (3) can-train-and-test-v1.5, (4) UNIMORE Bus-Off, (5) UNIMORE DAGA, and (6) UNIMORE Ventus. The two HCRL datasets are well-established in the literature, whereas can-train-and-test-v1.5 is a promising new dataset—and the three UNIMORE datasets lie somewhere in between. In our evaluation, we pit sixteen machine learning IDSs against each dataset and analyze the results. In addition, we conduct a feature evaluation of can-train-and-test-v1.5, and we

investigate the impact of train-test interdependence in the three UNIMORE datasets.

We find that, when pitted against the five comparison datasets, can-train-and-test-v1.5 paints a clearer picture of an IDS's true capabilities; in fact, can-train-and-test-v1.5's testing scenarios can reveal when an IDS has overfitted to a particular vehicle type—unlike the UNIMORE datasets. Furthermore, unlike the HCRL datasets, can-train-and-test-v1.5 provides more than enough data to train a complex machine learning model—an order of magnitude more—reducing the risk of underfitting. Moreover, can-train-and-test-v1.5 maintains ample differentiation power; the standard deviation of the models' F1-scores was 0.2392 (excluding suppress attacks), whereas the standard deviations for the remaining datasets—HCRL Car Hacking, HCRL Survival Analysis, UNIMORE Bus-Off, UNIMORE DAGA, and UNIMORE Ventus—were 0.2254, 0.2333, 0.1824, 0.2121, and 0.2100 (excluding suppress attacks), respectively.

Keywords Automotive · Controller Area Network · In-Vehicle Network · Intrusion Detection System · Machine Learning

Brooke Kidmose
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Kongens Lyngby, Denmark
E-mail: blam@dtu.dk

Andreas Kidmose
Department of Electrical and Photonics Engineering
Technical University of Denmark
Kongens Lyngby, Denmark
E-mail: abki@dtu.dk

Weizhi Meng
School of Computing and Communications
Lancaster University
Lancaster, United Kingdom LA1 4YW
E-mail: weme@dtu.dk

1 Introduction

This paper is an extension of *can-sleuth: Investigating and Evaluating Automotive Intrusion Detection Datasets*, originally presented at the 2024 European Interdisciplinary Cybersecurity Conference [54].

Modern automobiles rely on interconnected Electronic Control Units (ECUs) to manage critical systems—from engine control and fuel injection to transmission shifting and cruise control. Many of these safety-critical communications cross the Controller Area Network (CAN) bus, a

protocol designed in the 1980s with robust error detection and fault tolerance, but limited security features. While the original CAN specification assumed a closed system without external connectivity, today’s vehicles are increasingly connected—to the internet, to other vehicles, and to road infrastructure—creating new security challenges.

The automotive industry has evolved beyond controller area networks, adopting high-bandwidth protocols—such as automotive Ethernet—and implementing various security measures—such as gateway modules. However, the CAN bus remains fundamental to vehicle operations, particularly for safety-critical systems [126, 93]. While these added security measures help protect CAN networks by securing access points and network boundaries, the core CAN protocol retains its original limitations—most notably, its lack of native authentication and encryption. The protocol’s widespread deployment in safety-critical systems (in part due to regulatory requirements) combined with its inherent security limitations, presents an attractive target for cyberattacks (for more information, see Section 3). In an increasingly connected automotive ecosystem, these vulnerabilities pose risks beyond individual vehicles, potentially impacting vehicle-to-vehicle communications and intelligent transportation infrastructure.

To address these security challenges, researchers have proposed various defensive strategies. These include intrusion detection systems [65, 66, 152, 37], firewalls and filtering mechanisms [75, 76, 37, 28], honeypots [144, 76], dynamic arbitration [41, 65], and cryptographic solutions such as authentication and encryption [107, 149, 30, 140, 77, 6, 12, 127]. Developing and evaluating these security technologies requires both deep understanding of CAN bus operations and access to substantial amounts of CAN traffic data—millions of frames are generated during just minutes of vehicle operation [63]. This is particularly crucial for machine learning-based approaches, which demand extensive datasets for training and testing.

In 2018, *Hacking and Countermeasure Research Lab* (HCRL) at Korea University published the HCRL Car Hacking dataset [123, 130, 122], which was soon followed by the HCRL Survival Analysis dataset [33, 34]. These datasets—especially the HCRL Car Hacking dataset—became extremely popular with automotive security researchers.

Lampe and Meng, researchers at the Technical University of Denmark, identified several limitations of existing open access datasets. In 2023, they curated the `can-train-and-test` dataset to better meet the needs of automotive security researchers [64, 67, 60, 68]. In 2024, Kidmose and Meng extended `can-train-and-test` into `can-train-and-test-v1.5`, featuring two testing subsets and two new attacks [54, 61].

In 2019, the Web Engineering and Benchmarking Laboratory at the *University of Modena and Reggio Emilia*

(UNIMORE) published the UNIMORE Bus-Off dataset [131, 132], followed in 2022 by the Detecting Attacks to in-vehicle networks via n-Gram Analysis (DAGA) dataset [133, 134] and in 2024 by the Ventus dataset [98, 97].

The above-mentioned datasets are open-access and are available online. For detailed descriptions of the datasets, see Section 4, and for information on how to access the datasets, see Section 4.4.

This paper constitutes a deep dive into the HCRL Car Hacking dataset (`hcr1-ch`), the HCRL Survival Analysis dataset (`hcr1-sa`), `can-train-and-test-v1.5` dataset, the UNIMORE Bus-Off dataset (`bus-off`), the UNIMORE DAGA dataset (`daga`), and the UNIMORE Ventus dataset (`ventus`).

By conducting experiments, we “sleuth out” important characteristics of our input data—e.g., the presence (or absence) of the `timestamp` feature, the subdivision (or lack thereof) of the `data field` feature. Moreover, we show that, compared to the HCRL datasets, `can-train-and-test-v1.5` provides much more training and testing data (as well as specialized testing scenarios) while maintaining equivalent (if not greater) differentiation power.

Next, we show that, compared to the UNIMORE datasets, `can-train-and-test-v1.5` paints a clearer picture of an IDS’s true capabilities. For example, `can-train-and-test-v1.5`’s testing scenarios can expose an IDS that has overfitted to a particular vehicle type; none of the UNIMORE datasets can. In addition, `can-train-and-test-v1.5` demonstrates more differentiation power than the three UNIMORE datasets: when evaluated against `can-train-and-test-v1.5`, the standard deviation of the models’ F1-scores was between 0.0271 and 0.0568 greater than the standard deviations of the models’ F1-scores when evaluated against the UNIMORE datasets.

Lastly, we leverage the UNIMORE datasets to investigate the impact of train-test interdependence. For all three UNIMORE datasets, the attack traces were constructed out of a small number of attack-free traces—either by injecting or removing CAN messages. Accordingly, if we train an IDS on these attack-free traces, then the IDS will see the exact same traces—plus attacks—during testing. We subdivided each UNIMORE dataset into two sub-datasets: sub-dataset #1’s training data contains the attack-free traces used to generate the attack traces found in its testing data, whereas sub-dataset #2’s training data contains the attack-free traces that were *not* used to generate the attack traces found in its testing data. We find that train-test interdependence (i.e., the conditions of sub-dataset #1) significantly improves the performance of most of our machine learning models. In fact, for the `daga` dataset, eight of the sixteen models performed better, five of the sixteen performed equally well, and only three performed worse.

1.1 Research Questions

- **RQ1:** How does the presence of the `timestamp` feature and the format of the `data field` (intact or subdivided) impact the performance of different machine learning models?
- **RQ2:** How does the newly-published `can-train-and-test-v1.5` dataset compare to the well-established `hcrl-ch` and `hcrl-sa` datasets?
- **RQ3:** How does `can-train-and-test-v1.5` compare to newer datasets such as `bus-off`, `daga`, and `ventus`?
- **RQ4:** How does train-test interdependence impact the performance of different machine learning models?

We developed our research questions in an effort to address existing gaps in the automotive intrusion detection literature. Existing works have experimented with different combinations of features and different machine learning models; however, few studies analyze a *large* number of *diverse* machine learning models. We evaluated sixteen different machine learning models, including both traditional machine learning and deep learning as well as both supervised and unsupervised approaches. In addition, we sought to analyze the newly-published `can-train-and-test-v1.5` through the lens of comparison. This comparison, which involves six different datasets, is much more extensive than its predecessor (Kidmose and Meng’s [54]), which pitted `can-train-and-test` against just two datasets.

The separation of research questions **RQ2** and **RQ3** reflects a deliberate methodological choice designed to enhance both analytical clarity and comparative depth. While a comprehensive analysis of all datasets might initially seem the most thorough approach, such a strategy would necessitate fifteen distinct pairwise comparisons among the six datasets (HCRL Car Hacking, HCRL Survival Analysis, `can-train-and-test-v1.5`, UNIMORE Bus-Off, UNIMORE DAGA, and UNIMORE Ventus). This exhaustive comparison approach would not only consume considerable space but could potentially obscure key insights beneath an overwhelming volume of comparative data.

To address these challenges, we adopted a structured comparative framework centered on the `can-train-and-test-v1.5` dataset. This choice leverages the dataset’s unique position within the spectrum of automotive intrusion detection datasets, as it bridges characteristics of both the HCRL and UNIMORE dataset families. Among all datasets studied, only `can-train-and-test-v1.5` contains the same four attack types present in HCRL Car Hacking: (1) Denial of Service, (2) Fuzzing, (3) RPM Spoofing, and (4) Gear Spoofing. Additionally, like HCRL Survival Analysis, it contains CAN bus data from multiple vehicles—a characteristic unique to these two datasets. Moreover, `can-train-and-test-v1.5` incorporates suppress attacks, a feature it shares with the newer UNIMORE Bus-Off and UNIMORE

Ventus datasets but which is absent from the HCRL datasets. These overlapping characteristics make it an ideal reference point for comparative analysis.

The separation of these comparisons into distinct research questions serves both practical and analytical purposes. It enables focused, detailed examination of specific dataset characteristics while maintaining a manageable scope for analysis and presentation. Our comparative strategy allows us to maintain continuity with our previous research—Kidmose and Meng [54]—while simultaneously exploring new directions in dataset development.

To answer our research questions, we designed and executed a number of experiments. To address **RQ1**, we conducted a series of experiments *with* the `timestamp` feature, followed by a series of experiments *without* the `timestamp` feature. Similarly, to assess the impact of subdividing the `data field` feature, we conducted a series of experiments *with* subdivision, followed by a series of experiments *without* subdivision. Next, for **RQ2**, we evaluated all of our machine learning models against `hcrl-ch`, `hcrl-sa`, and `can-train-and-test-v1.5`. We conducted both point by point comparisons (analyzing each sub-dataset and testing subset *individually*) and holistic comparisons (analyzing each model’s *average* performance). To address **RQ3**, we evaluated all of our machine learning models against `can-train-and-test-v1.5`, `bus-off`, `daga`, and `ventus`, and, as before, we analyzed the results point by point and holistically. Finally, for **RQ4**, we divided the three UNIMORE datasets into two sub-datasets: one with significant train-test interdependence; one without.

1.2 Contributions

Our contributions are as follows:

1. To address research questions **RQ1**, **RQ2**, **RQ3**, and **RQ4**, we adapt and apply sixteen different machine learning models to six different automotive intrusion detection datasets.
2. Using our machine learning models, we evaluate the impacts of (1) including or excluding the `timestamp` feature and (2) subdividing or not subdividing the `data field` feature. By analyzing the performance of each machine learning model during each scenario, we determine the ideal scenario for each model.
3. We compare the newly-published `can-train-and-test-v1.5` dataset to two well-established open-access datasets—the HCRL Car Hacking and Survival Analysis datasets. We demonstrate that the `can-train-and-test-v1.5` dataset contains novel features not available in either of the HCRL datasets, and we highlight a number of unique insights the `can-train-and-test-v1.5` can provide.

4. We compare the `can-train-and-test-v1.5` dataset to three newer datasets—the UNIMORE Bus-Off, DAGA, and Ventus datasets. We demonstrate `can-train-and-test-v1.5`'s unique ability to expose IDSs that have overfitted to a particular vehicle type, and we highlight the greater authenticity of `can-train-and-test-v1.5`'s live, on-the-road attacks vs. the UNIMORE datasets' simulated attacks.
5. We examine the effect of train-test interdependence on the performance of our machine learning models using the three UNIMORE datasets. Our results demonstrate that when the attack traces used for testing were generated by adding attacks to the attack-free traces used for training (train-test interdependence), our models performed significantly better than when the attack traces were generated from attack-free traces *not* used for training.
6. We present a systematic vulnerability analysis of eleven vehicles spanning model years 2011-2024, featuring empirical evidence that CAN bus vulnerabilities persist in modern vehicles. We evaluate the implications of our findings in the context of vehicle longevity and the threat posed by nation-state actors.

1.3 New Contributions

This work substantially extends our previous conference paper with several major new contributions:

1. We have expanded our dataset analysis to include three additional automotive intrusion detection datasets—UNIMORE Bus-Off, UNIMORE DAGA, and UNIMORE Ventus—complementing our previous analysis of HCRL Car Hacking, HCRL Survival Analysis, and `can-train-and-test`.
 - (a) Our experimental methodology remains consistent across both original and new datasets, enabling direct performance comparisons. We apply the same testing protocols to maintain analytical rigor.
 - (b) We have thoroughly integrated the new datasets throughout the paper, including comprehensive coverage in the “Descriptions of Datasets” section, comparative analysis in “Comparison of Datasets,” preprocessing details in “Methodology,” and performance evaluation in “Discussion of Results.”
2. We present new research on train-test interdependence, including experimental evaluation and analysis, using one of the UNIMORE datasets as our test case.
3. We provide what we believe to be a comprehensive survey of all existing open-access CAN intrusion detection datasets, offering a valuable resource for researchers in this field.

4. We introduce a new “Preliminaries” subsection to our “Methodology” section. In the new subsection, we explain our preprocessing strategies in greater detail. In addition, we explain how each of the features in our feature evaluation was formatted—or reformatted—before it was given to our machine learning models as input.
5. We conduct an empirical vulnerability analysis of eleven vehicles from model years 2011-2024, demonstrating that CAN bus vulnerabilities persist in modern vehicles. This new analysis includes successful bidirectional communication with a 2024 model vehicle, revealing that fundamental architectural weaknesses remain despite recent security advances.

1.4 Paper Organization

The remainder of this paper is organized as follows: Section 2 outlines pertinent background & related work—in particular, the CAN bus, open-access CAN datasets, and previous evaluations of CAN datasets. Section 3 details our CAN bus vulnerability survey. Section 4 describes the `hcrl-ch`, `hcrl-sa`, `can-train-and-test-v1.5`, `bus-off`, `daga`, and `ventus` datasets in detail—and provides information on how to find and access the datasets. In Section 6, we explain our methodology; then, in Section 7, we present and discuss our results. In Section 8, we address limitations of our research as well as opportunities for future work. Section 9 concludes our work.

2 Background & Related Work

The rapid digitization of automotive systems has transformed vehicles from mechanical machines to complex, interconnected electronic networks. This section explores three critical aspects of automotive network security: Controller Area Network (CAN) bus infrastructure, the open-access datasets used to study and simulate CAN traffic, and the methodological approaches for evaluating these datasets, providing a comprehensive examination of the challenges and opportunities in understanding and securing modern vehicular communication systems.

2.1 About the CAN Bus

Modern vehicles have evolved from mechanical to electronic systems, built on a complex network of communication protocols centered around the decades-old CAN protocol. CAN remains ubiquitous in part due to regulatory requirements: in the United States, the CAN bus has been mandatory for emissions compliance since 2008, ensuring its continued use despite known security vulnerabilities [67].

Modern in-vehicle networks typically incorporate multiple communication protocols—Local Interconnect Network (LIN), FlexRay, automotive Ethernet, and Media Oriented System Transport (MOST)—creating a layered, interconnected ecosystem [52]. Within this hybrid architecture, CAN remains a critical protocol for safety-critical communications, bridging legacy and emerging automotive technologies (for more information, see Section 3).

Developed by Robert Bosch GmbH in 1983 and standardized as ISO 11898 in 1993, the CAN protocol was designed to address three core engineering challenges: low latency, high throughput, and reliability [18, 67]. Its innovative design dramatically simplified vehicle wiring by using a dual-wire serial bus where every Electronic Control Unit (ECU) connects to the same two-wire system, enabling efficient communication through a sophisticated arbitration scheme [7, 67].

CAN frames follow a structured communication protocol: each frame contains a start-of-frame marker, an arbitration identifier determining message priority, control bits, a data length code, a data field (zero to eight bytes), a cyclic redundancy checksum, an acknowledgment field, and an end-of-frame marker [66, 67]. While the protocol is standardized, implementations remain proprietary—manufacturers assign unique arbitration IDs and encode distinct data into CAN frames; as such, automotive research—especially when it comes to passenger vehicles—may not be generalizable [18].

Technological evolution has introduced more advanced CAN variants. In 2011, Bosch released Controller Area Network Flexible Data-Rate (CAN FD), which increased the bitrate from 1 Mbps to 8 Mbps [9, 109, 52]. Although CAN FD does not inherently improve security, its higher bandwidth could support future security enhancements [65]. Bosch is now developing CAN XL, targeting bitrates up to 20 Mbps, with improved scalability to potentially support advanced security features [65, 52].

11-bit Standard CAN vs. 29-bit Extended CAN. A critical distinction exists between 11-bit standard CAN and 29-bit extended CAN identifiers. Standard 11-bit CAN, commonly used in passenger vehicles, offers limited addressing space, while 29-bit extended CAN provides more extensive addressing typically used in heavy vehicles such as trucks and buses. Both formats have demonstrated vulnerabilities, but research has predominantly focused on 11-bit CAN [8].

Extended CAN, particularly when implemented via the SAE J1939 standard, presents heightened security risks. The standard’s comprehensive specifications simplify attack development, potentially enabling reconnaissance and exploitation across a variety of vehicles and industries. Unlike 11-bit CAN attacks—which often require customization based on specific make, model, and year—J1939-compliant

29-bit CAN attacks can potentially generalize to any J1939-compliant vehicle from semi-trucks to farm tractors to city buses to school buses to garbage trucks [52, 8].

Researchers have demonstrated the practical implications of these vulnerabilities. Burakova et al. [8] successfully executed safety-critical attacks on heavy vehicles—a Class-8 2006 semi-truck and a 2001 school bus—using publicly available SAE J1939 standard information [8, 150]. They demonstrated comprehensive engine control capabilities, including:

- Inhibiting vehicle acceleration
- Manipulating engine RPMs to potentially destructive levels
- Disabling engine braking at speeds below 30 miles per hour

Particularly concerning was the researchers’ ability to execute several of the attacks without modification across the two different vehicle types (semi-truck and bus).

To address these challenges, standards bodies are developing security improvements. SAE International is creating the CAN FD Network Security J1939-91C standard, which aims to introduce cryptographic authentication and encryption to ensure message confidentiality, integrity, and authenticity [87, 11, 65].

Automotive Diagnostics. The CAN bus provides the communication backbone for automotive diagnostic technologies, including On-Board Diagnostics (OBD) and Unified Diagnostic Services (UDS), enabling comprehensive monitoring, analysis, and maintenance of the complex electronic systems that control modern vehicles.

On-Board Diagnostics is a standardized self-diagnostic system mandated for all on-road vehicles in the United States, primarily designed to monitor vehicle emissions and engine performance. As a request-response protocol, OBD utilizes Parameter Identifiers (PIDs)—standardized codes defined by the SAE J1979 standard—to request specific data from a vehicle’s Electronic Control Units (ECUs). These PIDs enable practitioners to extract consistent vehicle status information regardless of vehicle make and model.

The OBD communication protocol operates through a specific addressing system on the CAN bus. External test equipment typically uses the header 0x7DF to send diagnostic requests, with ECUs typically responding via the header 0x7E8. PIDs are embedded in the data field and enable queries such as fuel tank level, vehicle speed, or engine coolant temperature.

In contrast, *Unified Diagnostic Services*, codified in ISO-14229, represents an alternative diagnostic communication protocol over CAN networks. While both UDS and OBD facilitate automotive diagnostics, they serve distinctly different purposes. UDS is a more comprehensive diagnostic tool primarily used by technicians at service stations, en-

abling in-depth interactions with a vehicle’s electronic control units beyond the emission-focused scope of OBD. The UDS protocol defines a specific range of identifiers from 0x700 to 0x7FF, which can potentially be targeted by diagnostic attacks to acquire sensitive information about vehicles or ECUs [73]. Researchers have documented various diagnostic attacks exploiting UDS, highlighting the potential vulnerabilities in these communication protocols [82, 83, 85, 84, 72, 74, 151].

2.2 Open-Access CAN Datasets

In the succeeding paragraphs, we review relevant related work, namely, existing open-access CAN datasets—both attack-free (i.e., “benign”) datasets and intrusion detection datasets (which contain attacks). Table 1 highlights six open-access attack-free CAN datasets, while Tables 2 and 3 describe eighteen open-access intrusion detection datasets (i.e., CAN datasets which contain attacks).

CAN Datasets (attack-free). Though our paper focuses on intrusion detection datasets, we include these attack-free datasets because they can provide invaluable supplemental data to intrusion detection datasets. For example, many intrusion detection datasets—including *hcrl-ch*, *bus-off*, *daga*, and *ventus*—contain data from only one vehicle. When a machine learning IDS is trained against one of these datasets, it can overfit to one vehicle type.

Suppose we are using the *daga* dataset to train and test a machine learning IDS. Since *daga* contains data from only one vehicle (a 2016 Volvo V40 Kinetic), the IDS could overfit to that specific vehicle. We would not be able to detect this type of overfitting during testing, either, since *daga* only has one vehicle available for both training and testing. However, if we were to add supplemental data from one of the datasets in Table 1 to *daga*’s training data, we could mitigate overfitting during the training phase. Additionally, if we were to add supplemental to *daga*’s testing data, we would have a chance to detect overfitting issues; an overfitted IDS would likely produce false positives when analyzing supplemental data from a different vehicle.

Moreover, some machine learning IDSs, e.g., some autoencoders, train exclusively on attack-free data in order to detect anomalies during the testing phase. By encoding and decoding data, autoencoders are trained to efficiently encode—or compress—data while minimizing the reconstruction error [65, 23]. An autoencoder-based IDS would train on normal CAN traffic until reconstruction error is minimized. During the testing phase, if the autoencoder encounters normal CAN traffic, the reconstruction error will be minimal. If, however, the autoencoder encounters attack traffic, the reconstruction error will be significant, and the IDS will report an attack [65]. Alkhatib et al. [2] leveraged

this strategy to detect attacks in automotive Ethernet networks.

To date, a limited number of publicly available datasets containing exclusively attack-free (benign) CAN bus traffic have been identified in the literature [67]. These datasets provide researchers—and machine learning models—with an essential baseline reference for normal CAN traffic, supporting empirical research, intrusion detection system development, and CAN protocol enhancements.

The 2017 CrySyS Lab CAN dataset (CrySyS CAN) [59] contains attack-free raw CAN traffic captured during specific driving scenarios—such as lane changes and speed adjustments. Notably, a separate tool—CAN log infector—was published alongside CrySyS CAN. The tool allows researchers to simulate attacks using the attack-free data. In contrast, the 2019 AEGIS Big Data Project Automotive CAN Bus dataset (AEGIS) [44] provides signal data and global positioning system (GPS) information, but no raw CAN data.

The Reverse Engineering CAN Bus dataset (ReCAN) [155, 154] and the University of Turku’s Heavy-Duty Truck CAN Bus dataset (Truck CAN) [142] both significantly contribute to automotive network research by providing 29-bit extended CAN data, a format predominantly used in commercial and heavy vehicles, such as buses, trucks, and tractors. ReCAN provides data from passenger vehicles and commercial trucks, while Truck CAN focuses exclusively on the Renault T520 6X2, a heavy truck. Collectively, they address a critical gap in existing CAN data resources (for a detailed explanation of 11-bit and 29-bit CAN arbitration identifiers, see Section 2.1).

Researchers investigating automotive diagnostics, CAN bus reconnaissance, and automotive reverse engineering will find both the HCRL CAN Signal Extraction and Translation dataset and the CAN-Modes dataset to be invaluable resources. The HCRL CAN Signal Extraction and Translation dataset (HCRL SET) [129] was specifically designed for signal extraction research. Comprising 40 CAN traffic logs, the dataset was collected by periodically sending OBD queries under controlled driving conditions. Each log file is uniquely identified by the PID used during traffic collection. The CAN-Modes dataset provides both CAN traffic data and OBD data from three different vehicle models: a Chevrolet Cruze, a Volkswagen Gol G6, and a Ford Fiesta.

Both datasets focus on vehicle diagnostic data, utilizing the standard OBD PID framework defined by SAE J1979. The datasets differ in data presentation: HCRL SET captures OBD responses within the raw CAN traffic, while CAN-Modes offers both raw CAN logs and dedicated OBD-specific files. The CAN-Modes dataset uniquely provides decoded OBD responses, enabling direct interpretation of vehicle parameters such as fuel tank level and vehicle speed

Table 1: CAN Datasets (attack-free)¹

Date ²	Name	Acronym	Data source ³	11-bit or 29-bit?	References
10 Oct. 2017	CrySyS Lab CAN dataset	CrySyS CAN	Vehicle in motion	11-bit	CrySyS Lab [59]
03 Jul. 2019	AEGIS Big Data Project Automotive CAN Bus dataset	AEGIS CAN	Vehicle in motion—passenger vehicle	N/A—signal data only	Kaiser, Stocker, and Festl [44]
21 Jul. 2020	HCRL CAN Signal Extraction and Translation dataset	HCRL SET	Vehicle in motion—2010 model	11-bit	Song and Kim [129, 128]
07 Feb. 2020	Reverse Engineering CAN Bus dataset	ReCAN	Vehicle in motion—Alfa Romeo Giulia Veloce, Opel Corsa, Mitsubishi Fuso Canter, Isuzu M55, Piaggio Porter Maxi	Both	Zago et al. [155, 154]
31 May 2021	Heavy-Duty Truck CAN Bus dataset	Truck CAN	Vehicle in motion—Renault T520 6X2	29-bit	University of Turku [142]
13 Dec. 2024	CAN-Modes dataset	CAN-Modes	Vehicle in motion—Chevrolet Cruze, Volkswagen Gol G6, Ford Fiesta	11-bit	Dos Santos Roque, Da Silva Alves, and de Freitas [112, 111, 110]

¹ In addition to our own analysis of these datasets, we synthesized details from the following sources: [67, 64, 146, 103, 143] and [152].

² When available, the “date” column contains the publication date of the earliest peer-reviewed article introducing the dataset. If no peer-reviewed article is available, we use the date of the earliest preprint or the date the dataset was uploaded or published.

³ When describing a data source, we use the term “vehicle in motion” to refer to a vehicle that was driving while the data was collected (e.g., on a public road, on a closed course, on a dynamometer). Data collected from a “vehicle in motion” is generally superior to data collected from a stationary vehicle, as a life-threatening attack would typically occur while the vehicle was in motion.

(for a detailed explanation of the OBD PID framework, see Section 2.1).

CAN Intrusion Detection Datasets. The advantages of enriching an intrusion detection dataset with supplemental data from attack-free CAN traffic (Table 1) can also be achieved through strategic combinations of multiple intrusion detection datasets (Tables 2 and 3). Such combinations provide ample normal traffic and allow practitioners to leverage diverse attack scenarios.

Each intrusion detection dataset contains a finite quantity of data collected from one or more vehicles (and/or testbeds). By combining two or more datasets, practitioners could train an IDS on multiple vehicles and expose the IDS—during training, testing, or both—to a wider range of attacks. To leverage multiple datasets, practitioners could consolidate two or more datasets into one superset—one superdataset—as a preprocessing step. Alternatively, practitioners could train an IDS on the first dataset, then the second dataset, and so on. Once the IDS was trained, they could test in the same manner—one by one.

Unfortunately, practitioners who wish to leverage multiple datasets to train and test an IDS will encounter some roadblocks. For example, many of the datasets in Tables 2 and 3 are unlabeled; that is, the individual samples are not labeled (e.g., “benign” and “attack,” or “0” and “1”). As such, practitioners who wish to use them for supervised learning will need to manually label them. Some datasets—e.g., the CrySyS dataset of CAN traffic logs containing fabrication and masquerade attacks (CrySyS Attack) [20, 19, 21] and

the Real ORNL Automotive Dynamometer CAN intrusion dataset (ROAD) [146, 145]—provide metadata to help practitioners identify where the attackers are; however, using this metadata to label the attacks is easier said than done. We downloaded and reviewed all the datasets listed in Tables 2 and 3 to determine which ones to use for the comparison and analysis presented in this paper. Ultimately, we chose to include only pre-labeled datasets. We leave labeling and evaluating the unlabeled datasets to future work (see Section 8.2).

Sami et al.’s ML350 CAN Bus dataset (ML350 CAN) [114, 113] is labeled; however, it lacks timestamps. We use the *timestamp* feature in our feature evaluation (Section 6.2) and as an input to all of our machine learning models. We wanted our comparison to be consistent—all datasets should include timestamps that our machine learning models can use during training and testing. Therefore, we opted not to use the ML350 CAN dataset.

While our comparative analysis focuses on the HCRL Car Hacking and HCRL Survival Analysis datasets, we examined a number of additional HCRL datasets for completeness, including the HCRL In-Vehicle Network Intrusion Detection Challenge, HCRL Attack & Defense, HCRL X-CANIDS, HCRL CAN-FD Intrusion, HCRL M-CAN Intrusion, and HCRL B-CAN Intrusion datasets.

The HCRL In-Vehicle Network Intrusion Detection Challenge dataset (HCRL IVN ID) [35] shares similarities with the HCRL Survival Analysis dataset discussed in Section 4.1, utilizing the same three vehicles (Chevrolet Spark,

Table 2: CAN Intrusion Detection Datasets^{1,2}

Date ³	Name	Acronym	Data source ⁴	Attack types	Labels ⁵	References
07 Jun. 2016	Simulated CAN Bus dataset	Sim CAN	Testbed	1	No	Kang and Kang [49, 48]
30 Aug. 2017	HCRL CAN dataset	HCRL CAN	Vehicle—Kia Soul ⁶	3	No	Lee, Jeong, and Kim [70, 69]
30 Aug. 2018	HCRL Car-Hacking dataset	HCRL CH	Vehicle in motion—Hyundai YF Sonata ⁷	4	Yes	Seo, Song, and Kim [123, 122]; Song, Woo, and Kim [130]
10 Oct. 2018	HCRL Survival Analysis dataset	HCRL SA	Vehicle in motion—2015 Chevrolet Spark, 2010 Hyundai YF Sonata, 2015 Kia Soul	3	Yes	Han, Kwak, and Kim [33, 34]
25 Sep. 2019	Bus-Off dataset	Bus-Off	Vehicle in motion—2016 Volvo V40 Kinetic	2	Yes	Stabili and Marchetti [131, 132]
14 Nov. 2019	TU Eindhoven CAN bus intrusion dataset v2	TU CAN v2	Testbed, Vehicle in motion—Opel Astra, Renault Clio	5	No	Dupont et al. [14]
22 Nov. 2019 ⁸	HCRL In-Vehicle Network Intrusion Detection Challenge dataset	HCRL IVN ID	Stationary vehicle—Chevrolet Spark, Hyundai Sonata, Kia Soul	4	Yes	Han, Kwak, and Kim [35]
23 Mar. 2020	Synthetic CAN Bus dataset	SynCAN	Testbed	5	No	Hanselmann et al. [36, 86]
03 Nov. 2020	ML350 CAN Bus dataset	ML350 CAN	Vehicle in motion—Mercedes ML350	2	Yes	Sami et al. [114, 113]
25 Feb. 2021	HCRL Attack & Defense Challenge dataset	HCRL A&D	Vehicle in motion—2020 Hyundai Avante CN7 ⁹	4	Yes	Kang et al. [45, 47, 46]
2022 ¹⁰	HCRL CAN-FD Intrusion Dataset	HCRL CAN-FD	Vehicle in motion—2021 Genesis G80 ¹¹	3	Yes	Kim [56]
2022 ¹⁰	HCRL M-CAN Intrusion Dataset	HCRL M-CAN	Vehicle in motion—Genesis G80 ¹¹	2	Yes	Kim [57]
2022 ¹⁰	HCRL B-CAN Intrusion Dataset	HCRL B-CAN	Vehicle in motion—Genesis G80 ¹¹	2	Yes	Kim [55]

¹ In addition to our own analysis of these datasets, we synthesized details from the following sources: [67, 64, 146, 103, 143] and [152].

² For datasets with raw CAN frames, only 11-bit standard identifiers are used; no publicly available CAN intrusion detection datasets using 29-bit extended CAN identifiers were found.

³ When available, the “date” column contains the publication date of the earliest peer-reviewed article introducing the dataset. If no peer-reviewed article is available, we use the date of the earliest preprint or the date the dataset was uploaded or published.

⁴ When describing a data source, we use the term “vehicle in motion” to refer to a vehicle that was driving while the data was collected (e.g., on a public road, on a closed course, on a dynamometer). Data collected from a “vehicle in motion” is generally superior to data collected from a stationary vehicle, as a life-threatening attack would typically occur while the vehicle was in motion.

⁵ If individual samples (i.e., individual CAN frames) are pre-labeled as “benign” or “attack,” then the dataset meets our “labels” criterion. A dataset that provides metadata to identify the attacks does not meet this criterion, since practitioners would need to label the data themselves.

⁶ We were unable to determine whether the vehicle was in motion or stationary.

⁷ Attack-free data was collected while the vehicle was in motion; attack data was collected while the vehicle was stationary [103, 146].

⁸ We could not find a publication date for the dataset. While the HCRL website references a journal article, it pertains to the Survival Analysis dataset rather than the In-Vehicle Network Intrusion Detection Challenge dataset. We have used the date of the final competition in the “In-vehicle Network Intrusion Detection” track [32], as we assume that once the competition was over, the dataset could be made public.

⁹ Some attacks were conducted on a vehicle in motion; others were either conducted while the vehicle was stationary or were simulated (due to safety concerns).

¹⁰ We could not find a publication date for the dataset. The attack-free driving data was collected in 2022.

¹¹ The attack-free data was collected from a real vehicle; the attack data was simulated from the attack-free data.

Hyundai Sonata, Kia Soul) and incorporating three identical attack types (DoS, fuzzing, malfunction) plus one additional attack type (replay). While HCRL IVN ID offers a larger sample size—over 9 million samples compared to under 2 million in HCRL Survival Analysis [35]—several limitations led us to select HCRL Survival Analysis for our research. First, HCRL IVN ID is limited to stationary ve-

hicle data [35], whereas HCRL Survival Analysis includes dynamic driving data [33, 34]. The distinction is significant: stationary data predominantly consists of null values (zero RPMs, zero speed) with minimal ECU activity, while dynamic driving data captures the full range of vehicle operations with all ECUs active, providing a more comprehensive representation of real-world conditions. Moreover,

Table 3: CAN Intrusion Detection Datasets (continued)^{1,2}

Date ³	Name	Acronym	Data source ⁴	Attack types	Labels ⁵	References
11 May 2022	Transmission-Resuming Time-Based Intrusion Detection System dataset	TTIDS	Vehicle in motion—two different vehicles	1	Yes	Lee et al. [72, 74]
13 Jul. 2022	Detecting Attacks to in-vehicle networks via n-Gram Analysis dataset	DAGA	Vehicle in motion—2016 Volvo V40 Kinetic ⁶	6	Yes	Stabili et al. [133, 134]
13 Oct. 2023	can-train-and-test dataset	CT&T	Vehicle in motion—2011 Chevrolet Impala, 2011 Chevrolet Traverse, 2016 Chevrolet Silverado, 2017 Subaru Forester ⁷	9	Yes	Lampe and Meng [64, 67, 60, 68]
24 Oct. 2023	HCRL X-CANIDS Dataset (In-Vehicle Signal Dataset)	HCRL X-CANIDS	Vehicle in motion—2017 Hyundai LF Sonata e-VGT ⁶	5	Yes	Jeong et al. [43, 42]
15 Dec. 2023	CrySyS dataset of CAN traffic logs containing fabrication and masquerade attacks	CrySyS Attack	Vehicle in motion—2006 model ^{6,8,9}	2	No	Gazdag, Ferenc, and Buttyán [20, 19, 21]
22 Jan. 2024	Real ORNL Automotive Dynamometer CAN intrusion dataset	ROAD	Vehicle in motion—mid-2010s model	10 ¹⁰	No	Verma et al. [146, 145]
26 Feb. 2024	CAN-MIRGU dataset	CAN-MIRGU	Vehicle in motion—2016 electric vehicle	7-18 ¹¹	Yes	Rajapaksha et al. [106, 104, 105]
14 May 2024	Ventus dataset	Ventus	Vehicle in motion—2016 Volvo V40 Kinetic ⁶	2	Yes	Pollicino, Stabili, and Marchetti [98, 97]
06 Jun. 2024	can-train-and-test-v1.5 dataset	CT&T v1.5	Vehicle in motion—2011 Chevrolet Impala, 2011 Chevrolet Traverse, 2016 Chevrolet Silverado, 2017 Subaru Forester ⁷	11	Yes	Kidmose and Meng [54, 61, 64, 67]

¹ In addition to our own analysis of these datasets, we synthesized details from the following sources: [67, 64, 146, 103, 143] and [152].

² For datasets with raw CAN frames, only 11-bit standard identifiers are used; no publicly available CAN intrusion detection datasets using 29-bit extended CAN identifiers were found.

³ When available, the “date” column contains the publication date of the earliest peer-reviewed article introducing the dataset. If no peer-reviewed article is available, we use the date of the earliest preprint or the date the dataset was uploaded or published.

⁴ When describing a data source, we use the term “vehicle in motion” to refer to a vehicle that was driving while the data was collected (e.g., on a public road, on a closed course, on a dynamometer). Data collected from a “vehicle in motion” is generally superior to data collected from a stationary vehicle, as a life-threatening attack would typically occur while the vehicle was in motion.

⁵ If individual samples (i.e., individual CAN frames) are pre-labeled as “benign” or “attack,” then the dataset meets our “labels” criterion. A dataset that provides metadata to identify the attacks does not meet this criterion, since practitioners would need to label the data themselves.

⁶ The attack-free data was collected from a real vehicle; the attack data was simulated from the attack-free data.

⁷ Some attacks were conducted on a vehicle in motion; others were either conducted while the vehicle was stationary or were simulated (due to safety concerns).

⁸ The metadata describes the vehicle as a “2006 middle class vehicle” [21].

⁹ The fabrication attack was simulated via a physical testbed; the masquerade attack was simulated via software [20, 19, 21].

¹⁰ 33 attack traffic captures; 10 unique attacks—a fuzzing attack, four different spoofing attacks, four different masquerade attacks, and an accelerator attack [67].

¹¹ Attack types can be categorized broadly (7 types: DoS, fuzzing, replay, spoofing, suspension, masquerade, and multiple) or granularly (18 types when counting unique spoofing variants) [106, 104, 105].

we would generally expect life-threatening cyberattacks to occur while the vehicle is in motion; turning the steering wheel 180 degrees while the vehicle is idling would not endanger anyone, but at 75 miles per hour, it could be a death sentence. Additionally, HCRL IVN ID’s documentation presents challenges; attack types are clearly labeled in “train” directories but not in “release” directories, which use generic nomenclature (e.g., “File.1.csv”). Given these considerations, we opted for the HCRL Survival Analysis

dataset, which provides comprehensive documentation both on the HCRL website and in a peer-reviewed journal article.

The HCRL Attack & Defense dataset (HCRL A&D) [45, 47, 46] contains all of the data features we require, namely, (1) timestamp, (2), arbitration ID, (3) data field, and (4) labels. However, it is quite similar to the HCRL Car-Hacking dataset and the HCRL Survival Analysis dataset—which we already included in our comparison. HCRL A&D includes

the following attacks: (1) flooding, (2) fuzzing, (3) replay, and (4) spoofing. These attacks overlap the attacks available in `hcr1-ch` and `hcr1-sa` (see Section 4.1). Moreover, HCRL A&D’s attack-free data is quite limited [103]; only two files/traces contain purely attack-free traffic—`Pre_train_S_0.csv` and `Pre_train_D_0.csv`, (180,687 lines and 179,347 lines, respectively) [45, 47, 46]. The 360,034 attack-free samples might not be enough to properly train a machine learning model—a problem shared by `hcr1-sa` (see Section 4.1).

As with HCRL A&D, the HCRL X-CANIDS dataset [43, 42] contains all of the required data—timestamps, arbitration IDs, data fields, and labels. It provides both raw CAN data and signal data. *Signals* are meaningful pieces of information encoded into CAN messages. Signal data is easier for human practitioners to understand and leverage, and it can prove highly beneficial to IDSs. The AEGIS Big Data Project Automotive CAN Bus (AEGIS CAN) dataset [44], the Reverse Engineering CAN Bus (ReCAN) dataset [155, 154], the Synthetic CAN Bus (SynCAN) dataset [36, 86], and the ROAD dataset [146, 145] all provide signal data (note that signal data can vary significantly in format and degree of “decoding”) [67].

In this paper, we limit our comparison to raw CAN data, not signal data. Hence, when we reviewed HCRL X-CANIDS, we focused on the raw CAN data stored in Apache Parquet files. Similar to the UNIMORE Bus-off, DAGA, and Ventus datasets (see Section 4.3), the attack traces in the HCRL X-CANIDS were generated by synthetically adding attacks to attack-free data. Jeong et al. [43, 42] explain that they conducted attacks during the period 480-1440 (half the length of the capture), to create “label-balanced” data. However, in a realistic scenario, the data would not be balanced; in fact, it would be extremely imbalanced. An hour of driving will generate several million CAN messages. Drivers spend hours upon hours on the road—all without incident. Thus, an automotive IDS should be designed with extreme class imbalance in mind. In this regard, the DAGA and Ventus datasets—which are already included in our comparison—are much more realistic (see Section 5.3).

Three HCRL datasets were generated from a Genesis G80 vehicle driven in loops around Korea University in 2022: the CAN-FD Intrusion dataset (HCRL CAN-FD) [56], the M-CAN Intrusion dataset (HCRL M-CAN) [57], and the B-CAN Intrusion dataset (HCRL B-CAN) [55]. Data for the HCRL CAN-FD dataset was collected during a dedicated driving session, whereas HCRL M-CAN and HCRL B-CAN share data from a single (approximately) one-hour driving sequence. All three datasets maintain consistent field structures—timestamp, arbitration ID, data length code, data field—though they differ in labeling

conventions: HCRL CAN-FD denotes normal/attack messages with R/T while the others use 0.0/1.0.

The HCRL CAN-FD dataset particularly notable as the first publicly available CAN intrusion detection dataset featuring the Controller Area Network Flexible Data-Rate (CAN FD) protocol. Introduced by Bosch in 2011, CAN FD enhanced the original protocol by increasing the bitrate from 1 Mbps to 8 Mbps and expanding the data field capacity from 8 to 64 bytes [9, 109, 52] (for more information about CAN FD, see Section 2.1). While HCRL CAN-FD includes three attack types—flooding (i.e., DoS), fuzzing, and malfunction—it notably lacks an attack-free trace, limiting its utility for IDS training and development. In contrast, HCRL M-CAN and HCRL B-CAN provide both attack-free and attack traces, though they implement only DoS and fuzzing attacks. Although all three datasets were collected from a Genesis G80, only HCRL CAN-FD captures CAN FD data, presumably because CAN FD is implemented almost exclusively on high-priority, safety-critical CAN subnetworks.

HCRL M-CAN and HCRL B-CAN focus on specific CAN subnetworks: M-CAN handles navigation and multimedia systems [57], while B-CAN manages low-priority functions such as body control module (BCM) lights, power windows, and smart key modules. These specialized datasets were excluded from our analysis for two reasons: first, they represent isolated subnetworks rather than comprehensive vehicle communications, and second, they primarily cover non-safety-critical systems where security breaches, while problematic, pose limited physical risk. Additionally, the CAN FD protocol exceeds the scope of our research, as our machine learning models were designed for classical CAN’s 8-byte data fields, and cross-protocol comparisons could introduce confounding variables in our analysis.

Lee et al.’s Transmission-Resuming Time-Based Intrusion Detection System dataset (TTIDS) [72, 74] represents a significant advancement in automotive security research as, to our knowledge, it is the first to provide CAN bus data from masquerade attacks on real vehicles using bus-off and Unified Diagnostic Services (UDS) attacks [73]. Unlike previous datasets that relied on programmatic simulation, such as `can-train-and-test-v1.5` [54, 61, 64, 67], TTIDS is composed of real attacks on real vehicles: Lee et al. [72] suspended specific Electronic Control Units (ECUs) and injected malicious CAN messages at the original transmission frequency [73].

While this approach offers higher fidelity than simulated attacks, several limitations warrant consideration. The dataset focuses exclusively on masquerade attacks, and the vehicle’s operational state during data collection is not specified in the documentation. However, given that diagnostic sessions typically require vehicles to be either stationary or moving below ten miles per hour [82]—with many

ECUs refusing to enter diagnostic mode while the engine is running [85]—it is reasonable to assume the diagnostic masquerade attacks were conducted on a stationary vehicle. This assumption, if true, leads to fidelity trade-offs: while attacks on real vehicles provide more authentic data, they may not fully represent attacks during actual driving conditions, which simulated attacks (on driving traces) can approximate. As demonstrated by Miller and Valasek [85, 84], a complete real-world attack typically proceeds in phases: first initiating a diagnostic session while the vehicle is stationary or moving slowly, then beginning the ECU reprogramming process (which effectively disables the ECU by erasing its firmware), and finally executing the masquerade attack as the vehicle resumes normal operation. Since Lee et al. do not mention reprogramming and permanently disabling ECUs [72, 73], we assume they captured only the stationary phase. The utility of the TTIDS dataset is further limited by data availability issues: the version linked in the paper (accessible via Google Forms) contains only attack traffic [71], while the GitHub repository provides only attack-free idling data, despite documentation indicating that attack-free driving data should be included [74].

The CAN-MIRGU dataset [106, 104, 105] provides a comprehensive collection of attacks on a 2016 electric vehicle. The dataset includes multiple attack types: denial of service (DoS), fuzzing, replay, spoofing, suspension (i.e., “suppress”), masquerade, and multiple attacks. While some attacks produced minimal effects, others, such as the gear shifting attacks, had serious consequences, such as stiffening or loosening the steering wheel. These attacks were conducted with the autonomous driving mode deactivated, which may limit the dataset’s real-world representativeness.

CAN Voltage Datasets. While beyond the primary scope of our investigation, two notable datasets—the SIMPLE dataset [17, 16] and the ECUPrint dataset [100, 29, 99]—offer valuable CAN voltage measurements that may be of interest to researchers developing multi-layer intrusion detection systems. A multi-layer IDS analyzes multiple communication layers simultaneously, detecting anomalies in both CAN message content and physical voltage characteristics to identify attacks that might escape single-layer monitoring.

The SIMPLE dataset, collected by Foruhandeh et al., captures CAN bus voltage samples from two vehicles—a 2011 Subaru Outback and a 2016 Nissan Sentra—using a Tektronix DPO 3012 oscilloscope connected via the On-Board Diagnostics port [17, 16]. These voltage measurements have been utilized in research on physical-layer intrusion detection, including studies on ECU identification and masquerade attack detection [125]. The ECUPrint dataset provides an even more comprehensive collection, featuring voltage and clock skew data from 54 Electronic Control Units (ECUs) across ten vehicles, including nine cars and

one tractor [100, 29, 99]. By capturing physical characteristics such as mean voltage, maximum voltage, clock skew, and plateau time, these datasets offer researchers unique insights into potential physical-layer CAN intrusion detection methodologies that extend beyond traditional CAN message-based analysis.

2.3 Evaluations of CAN Datasets

This section reviews existing literature evaluating open-access CAN intrusion detection datasets, focusing on comparative analyses and the development of assessment criteria.

In 2019, Al-Jarrah et al. [90] conducted a systematic review of intrusion detection systems (IDSs) for in-vehicle networks (IVNs), analyzing 42 research works in the field. Their findings revealed that, when evaluating proposed IDSs, 50% of studies leveraged real vehicle data, 26% relied on simulated data, and 24% failed to acknowledge their data sources. Regarding attack types, 69% of works focused on replay and/or message injection attacks, 26% on message tampering, 17% on DoS, 10% on malware attacks injecting malicious messages into the CAN bus, 7% on masquerade attacks, and 29% on others (or no attack types in particular). The survey highlighted a significant gap: the lack of publicly available CAN intrusion detection benchmark datasets for automotive IDS research.

Wu et al. [152] provided a complementary survey focusing on IDS approaches and available datasets. They classified 20 different in-vehicle IDSs into four categories: fingerprinting, parameter monitoring, information-theoretic, and machine learning. They found that machine learning-based IDSs outperformed non-learning IDSs at detecting unknown attacks but faced computational challenges. At the time of Wu et al.’s publication, just two open-access CAN intrusion detection datasets were widely known and available: the Simulated CAN Bus dataset (Sim CAN) [49, 48] and the HCRL CAN dataset [70, 69]. Notably, both datasets are unlabeled, significantly limiting their utility for IDS applications, particularly the machine learning approaches that showed promise in detecting novel attacks.

In 2021, Swessi and Idoudi [135] conducted a comparative analysis of datasets for both in-vehicle and inter-vehicle networks, examining their suitability for IDS development and evaluation. They established eleven fundamental metrics for assessing automotive intrusion detection datasets:

- Availability
- Large volume of data
- Real traces
- Labeled data
- Feature set
- Diversity of attacks

- Diversity of vehicles
- Attack severity
- Complete traffic
- Complete documentation
- Network configuration

Based on these criteria, they recommended the HCRL Survival Analysis dataset (HCRL SA) [33, 34] and the Real ORNL Automotive Dynamometer CAN intrusion dataset (ROAD) [146, 145]. However, they noted limitations of both: HCRL SA demonstrated low attack severity, while ROAD lacked vehicle diversity and normal/attack labels. Multiple researchers [103, 146, 73] have since uncovered additional deficiencies in HCRL SA, including inconsistencies in attack implementations and documentation discrepancies, ultimately challenging its suitability for meaningful intrusion detection system evaluation.

Karopoulos et al. [50] conducted a meta-analysis of existing surveys on in-vehicle network intrusion detection, proposing a unified IDS taxonomy based on location, type, layering, and reaction type. The authors emphasized the critical role of datasets and simulators in IDS development, as these resources provide the foundational data necessary for training and validation. Their analysis of available datasets highlighted a significant challenge: while CAN datasets are numerous, they lack diversity in vehicle makes and models, limiting their utility for developing robust IDSs.

The authors review several of the datasets discussed in Section 2.2, though their dataset naming conventions often diverge from established literature. For instance, their designation of the HCRL Attack & Defense Challenge dataset [45, 47, 46] as “Car Hacking Dataset v2” contradicts both the official naming on HCRL’s website and IEEE Dataport (“Car Hacking: Attack & Defense Challenge 2020 Dataset”) and common literature citations [103, 73, 67]. Their analysis also contains technical inaccuracies, such as misidentifying the CL2000 CAN data logger as a Mercedes model when discussing the ML350 CAN Bus dataset [114, 113].

Karopoulos et al. concluded that while CAN intrusion detection datasets were abundant, they had two fundamental limitations: (1) lack of diversity—with respect to network technology—and (2) inadequate data volume—especially for the development and evaluation of machine learning IDSs. The authors offer no specific recommendations regarding publicly available CAN intrusion detection datasets.

In 2022, Vahidi, Rosenstatter, and Mowla [143] conducted a comprehensive analysis of data readiness across both in-vehicle and inter-vehicle network datasets. They developed quantifiable metrics organized into hierarchical bands ranging from C to AAA, with weighted scoring criteria. Their evaluation framework encompasses sixteen metrics distributed across four bands:

- C

- Dataset documentation
- Objective
- Parseability
- Dataset age
- B
 - Format correctness and consistency
 - Dataset size
 - Completeness
 - Label inclusion and correctness
- A
 - Class balance
 - Attack documentation
 - Security coverage
 - Attack realism
- AA
 - Dataset realism and diversity
 - Feature context and documentation
 - Difficulty
 - Transformation and anonymization
- AAA
 - N/A

Their analysis revealed significant limitations in existing datasets. Notably, they demonstrated the potential brittleness of IDS systems trained on the HCRL Car Hacking dataset: when an IDS is trained against the HCRL Car Hacking dataset’s DoS attack trace, which uses an arbitration ID of “000,” and then is tested against *the same DoS attack trace* but with a slightly different high-priority arbitration ID, “002,” then performance is substantially degraded.

Among the evaluated IVN datasets, only one achieved Band A classification, with none reaching Bands AA or AAA. While specific dataset ratings were not disclosed, their assessment identified four critical shortcomings: (1) insufficient documentation regarding context, attack types, and attacker capabilities; (2) inadequate labeling that limits IDS applications; (3) unrealistic attack scenarios; and (4) unrepresentative data collection, typically limited to CAN bus traffic captured from diagnostic ports that may not reflect complete network behavior due to, e.g., gateways, firewalls, and proxies.

Two datasets mentioned in their survey, “Bi2022” [3] and “Hisingen,” are excluded from our analysis due to availability issues. The “Bi2022” dataset, formerly hosted at <http://49.232.218.41:8000/data.zip>, is no longer available, and extensive searches have not yielded alternative sources. The “Hisingen” dataset appears to originate from an unpublished project report. As our research focuses exclusively on publicly accessible CAN bus traffic datasets, these datasets fall outside our scope of analysis.

In 2023, Lee et al. [73] conducted a comprehensive evaluation of CAN intrusion detection datasets that closely parallels our research methodology. Their study assessed six machine learning models—four traditional (Naive

Bayes, Decision Tree, Stochastic Gradient Descent, Random Forest) and two deep learning models (Deep Convolutional Neural Network, Long Short-Term Memory)—against five distinct attack types sourced from multiple datasets. While their approach shares methodological similarities with ours, our analysis is more extensive, encompassing sixteen machine learning models evaluated against six different datasets, providing a broader comparative framework for assessing CAN intrusion detection datasets.

Lee et al.’s study encompassed two classes of attacks: masquerade attacks (diagnostic and bus-off variants) from the Transmission-Resuming Time-Based Intrusion Detection System dataset (TTIDS) [72, 74], and network attacks (DoS, fuzzing, and spoofing) likely sourced from the HCRL Car Hacking dataset [123, 122, 130] and/or the HCRL Attack & Defense Challenge dataset [45, 47, 46]. Their findings revealed significant performance disparities across attack types:

- DoS attacks were detected with near-perfect accuracy (F1-scores: 0.9999–1.0000).
- Fuzzing attacks saw moderate to high detection success (F1-scores: 0.4787–0.9890).
- Spoofing attacks were the most challenging to detect (F1-scores: 0.0000–0.0459).
- Masquerade attacks saw intermediate detection rates, with F1-scores of 0.1315–0.6674 for the diagnostic variant and 0.1506–0.7168 for the bus-off variant.

These findings noticeably diverge from previous research. Lampe and Meng’s survey [65] reported consistently high performance by similar IDSs pitted against the HCRL Car Hacking dataset (HCRL CH). For instance, Yang et al. [153] achieved average F1-scores of 0.999 on HCRL CH using Decision Tree and Random Forest models, while Alfaridus and Rawat [1] reported perfect detection (F1-score: 1.0) across all attack categories with a Random Forest model. This discrepancy is particularly significant given Rajapaksha et al.’s [103] analysis, which revealed that attacks in HCRL CH exhibit distinct arbitration ID frequency patterns, making them readily detectable using frequency- or sequence-based detection methods. The performance disparity might be attributed to Lee et al.’s potential use of the HCRL Attack & Defense Challenge dataset, either exclusively or in conjunction with HCRL CH, for the DoS, fuzzing, and spoofing attacks.

In addition to their comparative analysis, Lee et al. [73] established a comprehensive framework of qualitative and quantitative metrics for evaluating automotive intrusion detection datasets:

- Qualitative Metrics
 - **Dataset Documentation.** The dataset must include comprehensive documentation detailing its charac-

teristics and intended applications to enable proper assessment and utilization.

- **Dataset Collection Environment.** This metric evaluates data fidelity by distinguishing between synthetic (simulated) data collection and real-vehicle data collection methods.
- **Realistic Attack Dataset.** High-quality datasets prioritize attacks executed against actual vehicles with real-time data collection and verifiable vehicle impact, while simulated attacks must accurately replicate both CAN bus behavior and dynamic attack responses.
- **Labeled Dataset.** The dataset must provide accurate per-sample labeling, which is crucial for machine learning applications and model training.
- **Correctness and Consistency.** All data must be presented in a standardized format and demonstrate freedom from corruption.
- **Dataset Diversity.** The dataset should encompass varied operational conditions, including different driving scenarios, vehicles, and drivers, along with a comprehensive range of attack scenarios.
- Quantitative Metrics
 - **Attack Coverage.** This metric assesses dataset comprehensiveness by evaluating the presence of eight potential CAN bus attacks: DoS, fuzzing, replay, spoofing, diagnostic, bus-off, suspension, and masquerade attacks.
 - **Dataset Balance.** This metric evaluates the proportion of attack-free messages to attack messages within the dataset. While Lee et al. advocate for balanced datasets, we contend this approach may be problematic, as the extreme rarity of real-world automotive cyberattacks [52, 101, 115] suggests that training systems on artificially balanced data could lead to excessive false positives in deployment.
 - **Attack Complexity.** This metric assesses attack sophistication by prioritizing subtle attacks that closely mimic normal traffic patterns over easily detectable variants, thereby emphasizing the importance of detecting sophisticated threats.

Lee et al. highlight the fact that none of the datasets in their study provide data from multiple drivers, a limitation they contend remains unaddressed by existing open-access datasets. While this observation was accurate with regard to the datasets included in their study, it is worth noting that two more recent datasets have since addressed this constraint: `can-train-and-test` [64, 67, 60, 68] and its enhanced successor, `can-train-and-test-v1.5` [54, 61, 64, 67]. Both datasets deliberately incorporate demographic diversity by including CAN bus data from drivers of varying ages and sexes.

In 2024, Verma et al. [146] introduced the ROAD dataset and conducted a comprehensive evaluation of existing open-source automotive intrusion detection datasets. Their analysis revealed several inconsistencies in the HCRL dataset family, particularly in the HCRL CAN dataset developed by Lee, Jeong, and Kim [70, 69]. Notable discrepancies were found between the documentation available on the HCRL website, the dataset’s introductory conference paper, and the actual dataset contents. A significant example involves HCRL CAN’s implementation of a masquerade attack, termed “impersonation attack” by the authors. The documentation explicitly states:

“First, we assume an attacker who is capable of physical modifications can take control of a target node, stopping transmission of messages, and let impersonating node send messages to replace the terminated node. Since the malicious node supersedes the target node, the system can not detect changes of overall messages. We call this type of attack as impersonation attack.” [70, 69]

However, analysis of the dataset reveals that during the attack, the legitimate node continues to transmit messages alongside the malicious messages—more characteristic of a spoofing attack rather than the documented masquerade attack. Further inconsistencies were identified in the attack timing: while the documentation indicates that fuzzing and impersonation attacks commence at approximately 250 seconds, Verma et al.’s analysis [146] indicates that these attacks actually start at zero seconds. Similar documentation discrepancies were found in both the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset, aligning with our own findings, which are detailed in Section 4.1.

Lampe and Meng [67] identified the following critical limitations of existing open-access CAN intrusion detection datasets:

- **Lack of sufficient attack-free data.** Machine learning models, particularly deep learning architectures, require substantial attack-free data to establish reliable baseline behavior and minimize false positives.
- **Lack of sufficient attack variation.** Detection systems need exposure to a broad spectrum of attack variations, especially subtle intrusions, to develop robust detection capabilities, particularly when it comes to novel attacks.
- **Lack of sufficient vehicle variation.** Models must be trained and validated across different vehicle types to ensure generalization capabilities, as CAN implementations can vary significantly between manufacturers and models.
- **Lack of fidelity.** Datasets that poorly approximate real-world vehicle conditions may lead to models that fail to perform effectively in production environments, as they train on unrealistic or oversimplified scenarios.

- **Lack of severity.** Datasets should include high severity attacks to ensure that intrusion detection systems can identify potentially catastrophic intrusions, not just nuisance anomalies.
- **Lack of modernity.** Modern vehicles implement advanced security features such as gateways (as discussed in Section 3), which may limit the relevance of older attack strategies.
- **Lack of labels.** Comprehensive and accurate labeling is crucial for supervised learning approaches; practitioners often struggle to use—let alone label—unlabeled CAN datasets without intimate knowledge of the data collection and attack implementation methodologies.

Notably, of all the papers reviewed in this section (i.e., papers that evaluate open-access CAN intrusion detection datasets), only Lampe and Meng [67] discuss the UNIMORE Bus-Off, DAGA, and Ventus datasets—which we include in our comparative analysis later in this paper. This observation is particularly striking given that the Bus-Off dataset was released in 2019 and the DAGA dataset in 2022, while several of the dataset evaluation papers were published as recently as 2023 and 2024.

Guerra et al. [31] conducted a comparative analysis of machine learning model performance across three datasets: the ROAD dataset [146], the HCRL Car Hacking dataset [123, 122, 130], and the HCRL In-Vehicle Network Intrusion Detection Challenge dataset [35]—the latter sharing many of its characteristics with the HCRL Survival Analysis dataset. Their findings indicate that the ROAD dataset offers superior real-world representativeness through its incorporation of sophisticated, stealthy attacks. However, they identify two significant limitations: insufficient data volume for deep learning applications and severe class imbalance, particularly in Max Engine Coolant Temp fabrication and masquerade attacks. While the authors criticize this imbalance, we argue that such criticism may be misplaced—real-world automotive intrusion detection systems must operate in environments where attacks are exceptionally rare, creating even more extreme class imbalances than those present in current datasets. Therefore, training IDSs on artificially balanced datasets may inadequately prepare them for real-world deployment scenarios.

The evaluation frameworks developed by multiple researchers—Swessi and Idoudi [135], Vahidi, Rosenstatter, and Mowla [143], Lee et al. [73], and Lampe and Meng [67]—demonstrate significant convergence in their core assessment criteria. These frameworks consistently emphasize four fundamental metrics:

1. Dataset fidelity and realism
2. Attack severity and real-world applicability
3. Dataset diversity (including attack variations and vehicle types)

4. Label accuracy and completeness

3 CAN Bus Vulnerability Survey

While automotive security mechanisms have evolved significantly over the past two decades, substantial vulnerabilities persist in modern vehicles' controller area networks. Our empirical investigation of vehicles spanning model years 2011-2024 was shaped by several practical constraints: vehicle availability, duration of access, and owner comfort with active experiments. We obtained permission through three channels: direct consent from private vehicle owners, authorized access from an auto repair shop (with owner permission), and permission from an auto dealership for pre-sale vehicles. When evaluating vehicles operated by non-owners, we obtained driver consent as well.

The scope and depth of our experiments varied significantly based on these arrangements—some participants permitted only passive data collection, while others allowed more extensive security probing. Experimental duration also varied considerably: we spent months developing and refining non-destructive, visually verifiable attacks on three Chevrolet vehicles (2011 Impala, 2011 Traverse, and 2016 Silverado), while access to other vehicles was limited by business hours and our travel schedule. For instance, our investigation of the 2024 Buick Envista was constrained to a single evening session.

Through our experiments across multiple vehicle makes and models, we made an unsettling discovery: numerous modern vehicles continue to permit unauthorized CAN bus access. This vulnerability is concerning even when limited to reconnaissance capabilities (i.e., “receive only” access). While some manufacturers have implemented gateway security at the On-Board Diagnostics (OBD) port, the underlying CAN bus often remains accessible through various remote entry points. Security researchers have successfully demonstrated attacks by compromising both infotainment systems [82, 83, 25, 52] and remote services such as OnStar [58, 10, 24, 52]. In the absence of robust network segmentation between these systems and the safety-critical CAN bus, attackers who breach these auxiliary systems can potentially gain control over the vehicle's Electronic Control Units (ECUs). This architectural vulnerability means that even vehicles with well-secured diagnostic ports remain susceptible to attacks through other network interfaces.

These persistent architectural vulnerabilities have prompted automotive security researchers to propose three fundamental improvements to CAN bus security [52, 157, 121, 81]: (1) encryption to prevent unauthorized reconnaissance activities, (2) authentication and authorization to prevent unauthorized ECU manipulation, and (3) intrusion detection systems (IDSs) to provide defense-in-depth protection when primary security measures fail. This multi-layered

approach addresses both existing gateway vulnerabilities and the need for comprehensive security measures once encryption and authentication mechanisms are implemented.

Table 4 summarizes the empirical results of our CAN bus vulnerability survey, representing all vehicles we could obtain permission to test during our research period. While not comprehensive across all makes and models, our sample provides valuable insights into real-world vehicle security across different manufacturers and model years.

In our experiments with a 2024 Buick Envista, we successfully established bidirectional communication with the CAN bus via the OBD port, though we were unable to execute observable attacks during our limited testing window. This finding suggests that while modern vehicles may incorporate additional security layers, fundamental vulnerabilities in the CAN architecture remain exploitable.

The persistence of these vulnerabilities is particularly significant given the extended lifecycle of automotive assets. In 2024, the average age of vehicles in the United States reached 12.6 years [95, 141], with many vehicles remaining in service well beyond this average. This statistic has serious security implications: the two 2011 Chevrolet vehicles we tested, at 13 years old, represent the current average age of vehicles on American roads. Moreover, the 2016 Chevrolet Silverado and similar vehicles will likely remain in active service for many years to come. The CAN bus has been required by U.S. law since 2008 [52], creating a vast fleet of potentially vulnerable vehicles. Our testing demonstrated exploitable vulnerabilities in all three older vehicles—the 2011 Chevrolet Impala, 2011 Chevrolet Traverse, and 2016 Chevrolet Silverado—where we could both observe and manipulate CAN traffic with observable effects on vehicle behavior.

While some newer vehicles (2020-2023 models) demonstrated improved security measures, as evidenced by our inability to access any CAN data from the 2021 GMC Yukon, these protections are not universal. In fact, we were still able to establish bidirectional communication with a 2024 Buick Envista, though we could not confirm visible effects of our injected messages. Our ability to conduct reconnaissance on multiple 2017-2023 vehicles, including Subaru, Ford, and BMW models, demonstrates that reconnaissance and attack preparation remain feasible even in newer vehicles.

The threat landscape is further complicated by the hybrid nature of modern vehicle networks. While manufacturers have adopted automotive Ethernet for high-bandwidth applications—including infotainment systems and Advanced Driver-Assistance Systems (ADASs)—the CAN bus remains the primary protocol for safety-critical systems, driven by both its reliability and regulatory requirements. This architectural choice underscores the continued importance of CAN security research, as these networks directly control crucial vehicle safety systems.

Table 4: Empirical results of our CAN bus vulnerability survey¹

Vehicle Model	Year	Data Access	Attack Confirmation
Chevrolet Impala	2011	Bidirectional	Visually confirmed
Chevrolet Traverse	2011	Bidirectional	Visually confirmed
Chevrolet Silverado	2016	Bidirectional	Visually confirmed
Ford Focus	2017	Receive only ³	N/A
Subaru Forester (Red) ²	2017	Receive only ³	N/A
Subaru Forester (Gray) ²	2017	Receive only ³	N/A
Hyundai Kona	2020	No access	N/A
GMC Yukon	2021	No access	N/A
Chevrolet Silverado	2021	Receive only	N/A
BMW 120d	2023	Receive only ³	N/A
Buick Envista	2024	Bidirectional	Unconfirmed

¹ Active experiments were conducted with explicit permission from vehicle owners (private owners, repair shop clients, or dealerships) and, when applicable, the vehicles' operators.

² Two different vehicles of the same make, model, and year were evaluated (one red, one gray).

³ Our investigation was limited to data collection only (no active experiments) due to owner and/or driver restrictions.

The integration of these diverse networks, while necessary for features such as dashboard displays of acceleration, speed, etc., creates additional security risks. Compromised high-bandwidth systems could potentially serve as gateways to critical CAN networks. Moreover, vulnerabilities extend beyond the CAN bus itself—researchers have demonstrated successful attacks against other automotive protocols including Local Interconnect Network (LIN), FlexRay, and automotive Ethernet [108, 148, 156, 13, 52]. Only Media Oriented System Transport (MOST) has, to our knowledge, remained free from demonstrated vulnerabilities.

Recent trends in vehicle theft highlight critical vulnerabilities in automotive security systems. Sophisticated thieves have demonstrated the ability to steal vehicles by directly accessing the CAN bus through physical removal of body panels—often targeting areas where CAN wiring runs close to the vehicle's exterior, such as smart headlight assemblies. This approach circumvents gateway security measures typically present at the OBD port and infotainment system. By accessing the CAN bus behind these security gateways, attackers gain unrestricted network access. From this position, they can inject CAN frames that mimic legitimate key presence and trigger vehicle functions like door unlocking and engine start, enabling theft [139, 52]. While our CAN bus vulnerability survey did not include destructive physical access testing, these real-world exploits demonstrate that even modern vehicles remain vulnerable to attacks that bypass gateway protections through direct CAN bus access.

The dismissal of automotive attacks based on their physical access requirements or development complexity overlooks significant security implications. Miller and Valasek's pioneering research [82, 83, 85, 84] demonstrates that while exploit development requires substantial time and expertise, such barriers prove insufficient against determined adversaries. Their work suggests that nation-state actors, with

vastly greater resources, could achieve even more sophisticated capabilities. This concern is validated by historical precedent, e.g., the Stuxnet attack, which leveraged four zero-day vulnerabilities and is widely attributed to a US-Israel collaboration [4, 124]. Other nations with substantial cyber capabilities, such as Russia, have demonstrated their willingness to conduct large-scale attacks, as evidenced by numerous cyberattacks against Ukraine, including NotPetya [27, 26, 15].

Such actors could systematically acquire test vehicles, identify remote vulnerabilities, and launch synchronized attacks across entire vehicle populations—a capability Miller and Valasek demonstrated was possible through a vulnerability in Sprint's cellular network [82, 83, 25, 52]. This attack surface simply did not exist before manufacturers began treating vehicles as networked devices. The contrast with web security is striking: while we have largely abandoned unencrypted HTTP in favor of HTTPS, the automotive industry continues to rely on unencrypted CAN bus communications, leaving vehicles unnecessarily vulnerable to both physical and remote attacks.

This analysis, combined with our extensive vulnerability survey spanning vehicle makes and models from 2011-2024, reveals a concerning pattern: while automotive security has evolved, fundamental vulnerabilities persist in CAN bus architecture. The demonstrated feasibility of both physical and remote attacks, coupled with the potential for nation-state level operations, underscores the fact that securing access points alone is insufficient. Rather than relying on perimeter defenses, we must fundamentally harden the CAN bus protocol itself through encryption, authentication, intrusion detection systems, and defense-in-depth strategies.

Table 5: HCRL Car Hacking Dataset (hcr1-ch)^{1,2}

Sub-dataset	# of training samples	# of testing samples	Total # of samples
Attack-free	988,872	N/A	988,872
DoS	2,665,774	1,000,000	3,665,774
Fuzzing	2,838,863	1,000,000	3,838,863
Gear Spoofing	3,443,145	1,000,000	4,443,145
RPM Spoofing	3,621,705	1,000,000	4,621,705
TOTAL	13,558,359	4,000,000	17,558,359

¹ We exclude samples in duplicate CAN traces.

² We partitioned each attack capture into a training subset and a testing subset. We saved 1,000,000 CAN frames for the evaluation—approximately 25%.

Table 6: can-train-and-test-1.5^{1,2}

Sub-dataset	# of training samples	# of testing samples	Total # of samples
set_01	11,460,705	44,122,287	55,582,992
set_02	18,055,233	49,854,383	67,909,616
set_03	12,944,293	43,515,501	56,459,794
set_04	10,903,262	50,099,089	61,002,351
TOTAL	53,363,493	187,591,260	240,954,753

¹ We exclude samples in duplicate CAN traces.

² In [54] (Table 2), Kidmose and Meng analyzed can-train-and-test; in this paper, we analyze can-train-and-test-v1.5, which contains two new testing subsets. As such, our numbers differ from those reported in [54].

Table 7: HCRL Survival Analysis Dataset (hcr1-sa)^{1,2}

train	test_01	test_02	test_03	test_04	test_05	test_06	TOTAL
627,264	399,539	186,237	306,089	79,788	N/A ³	N/A ³	1,598,917

¹ We exclude samples in duplicate CAN traces.

² We curated hcr1-sa to match can-train-and-test-v1.5. We omitted one file: the attack-free file for the Chevrolet Spark, which we selected as our “unknown” vehicle. The omitted file contains 136,935 samples.

³ hcr1-sa lacks suppress and masquerade attacks; therefore, we were unable to create testing subsets test_05_suppress and test_06_masquerade.

Table 8: can-train-and-test-v1.5 (sub-dataset #1)¹

train	test_01	test_02	test_03	test_04	test_05	test_06	TOTAL
11,460,705	5,702,678	6,447,925	8,635,287	13,220,567	3,383,276	6,732,554	55,582,992

¹ We exclude samples in duplicate CAN traces.

Table 9: can-unimore-curated-v3 (detailed)^{1,2}

Dataset	Sub-dataset	# of training samples	# of testing samples	Total # of samples
bus-off	set_01	18,206,709	22,944,893	41,151,602
bus-off	set_02	17,351,795	21,962,749	39,314,544
bus-off	TOTAL	35,558,504	44,907,642	80,466,146
daga	set_01	14,947,920	17,747,506	32,695,426
daga	set_02	15,425,728	16,986,833	32,412,561
daga	TOTAL	30,373,648	34,734,339	65,107,987
ventus	set_01	27,270,266	28,531,914	55,802,180
ventus	set_02	26,449,101	28,195,576	54,644,677
ventus	TOTAL	53,719,367	56,727,490	110,446,857
All	TOTAL	119,651,519	136,369,471	256,020,990

¹ We exclude samples in duplicate CAN traces.

² can-unimore-curated-v3 denotes our second and final attempt to significantly resize and reorganize the Bus-Off, DAGA, and Ventus datasets to minimize errors (e.g., “out of memory”) during machine learning. Each of these datasets is split into two sub-datasets (as shown).

4 Description of Datasets

In this section, we describe the six datasets selected for our comparison, namely, the HCRL Car Hacking dataset (`hcr1-ch`), the HCRL Survival Analysis dataset (`hcr1-sa`), `can-train-and-test-v1.5` dataset, the UNIMORE Bus-Off dataset (`bus-off`), the UNIMORE DAGA dataset (`daga`), and the UNIMORE Ventus dataset (`ventus`).

For all six datasets, when calculating the number of samples, we exclude samples that are duplicated in `train_01.attack_free` and `train_02.with_attacks`. Thus, the “Total # of samples” column indicates the total number of samples in unique (non-duplicated) CAN traces.

4.1 The HCRL Car-Hacking and Survival Analysis Datasets

HCRL Car-Hacking dataset (`hcr1-ch`). Collected and curated by the Hacking and Countermeasure Research Lab (HCRL) at Korea University, the `hcr1-ch` dataset [123, 130, 122] contains attack-free CAN traffic as well as four types of attacks:

1. Denial of Service
2. Fuzzing
3. RPM Spoofing
4. Gear Spoofing

During the *denial of service* attack, messages with the arbitration ID “000” were injected at a high frequency. “000” is the highest-priority arbitration ID; as such, while “000” is transmitting, legitimate messages with lower-priority arbitration IDs are forced to wait. For the *fuzzing* attack, CAN frames were randomly generated (random arbitration ID, random data field) and injected. Lastly, for the *spoofing* attacks, a particular arbitration ID tied to a particular component or functionality was targeted (i.e., RPM gauge, gear). Only the target arbitration ID was injected.

All traffic was collected from a Hyundai YF Sonata, and all attacks were real, not simulated. According to Rajapaksha et al. [103], when it comes to automotive intrusion detection—and, in particular, the CAN bus—the HCRL Car-Hacking dataset is the most widely used. Researchers leverage `hcr1-ch` to develop, optimize, and evaluate automotive IDSs.

For the purposes of our comparison, we partitioned each attack capture into a training subset and a testing subset. We saved 1,000,000 CAN frames for the evaluation—approximately 25%.

Table 5 provides a numeric breakdown of the `hcr1-ch` dataset (compare to Tables 6 and 9). As shown in the table, the dataset contains 988,872 attack-free samples—enough to adequately train a machine learning model [78, 103] and significantly more than `hcr1-sa`.

However, Rajapaksha et al. [103] and Verma et al. [146] point out several issues with `hcr1-ch`. For example, the attack-free data was collected under driving conditions, whereas the attack data was collected while the vehicle was stationary (Verma et al. [146] leveraged signal decoding to confirm that the vehicle was not being driven when the attack data was collected). Verma et al. point out that the attack-free data—the training data—fundamentally differs from the attack data—the testing data—because the vehicle was driven for the former but not driven for the latter. Rajapaksha et al. [103] add that the attacks are too obvious; they reviewed a number of works in which `hcr1-ch` was used to evaluate an IDS, and they found that most IDSs achieved a 0.99+ F1-score for all attacks. Lastly, the dataset is formatted inconsistently; the attack-free data is stored in space-delimited `.txt` files, while the attack data is stored in comma-delimited `.csv` files.

HCRL Survival Analysis dataset (`hcr1-sa`). The `hcr1-sa` dataset [33, 34], also developed by the Hacking and Countermeasure Research Lab, contains data from three different vehicles—a Chevrolet Spark, a Hyundai YF Sonata, and a Kia Soul. In addition to attack-free CAN traffic, the dataset provides three types of attacks:

1. Flooding (i.e., Denial of Service)
2. Fuzzing
3. Malfunction (i.e., Spoofing)

The *flooding* attack is actually a denial of service attack. As in `hcr1-ch`, messages with the highest-priority arbitration ID, “000,” were injected at high frequency. The *fuzzing* attacks in `hcr1-ch` and `hcr1-sa` followed similar methodologies. However, the attack labeled “malfunction” warrants closer examination. Our analysis revealed that the “malfunction” attack is essentially a spoofing attack with some vehicle-specific variations:

- For the Chevrolet Spark and the Hyundai Sonata, the target arbitration IDs were injected with spoofed data fields that never varied throughout the trace.
- For the Kia Soul, the target arbitration ID was injected with a randomized data field; the attack was more characteristic of a fuzzing attack than the documented “malfunction” attack.

There is a notable discrepancy in the documentation regarding the Hyundai Sonata implementation. While the documentation states:

“The malfunction attack targets a selected CAN ID from among the extractable CAN IDs of a certain vehicle. As CAN IDs for the malfunction attack, we chose 0x316, 0x153 and 0x18E from the HYUNDAI YF Sonata, KIA Soul, and CHEVROLET Spark vehicles, respectively” [34].

The actual attack traces show that two arbitration IDs (0x316 and 0x43F) were targeted for the Sonata, not just one as implied.

For the purposes of our comparison, we curated `hcr1-sa` to match `can-train-and-test-v1.5`. `hcr1-sa` is partitioned into two training subsets (one attack-free subset, one attack subset) and four testing subsets (e.g., `test_01_known_vehicle_known_attack`), similar to the `can-train-and-test-v1.5` dataset (which contains the two original training subsets, the four original testing subsets, and two new testing subsets). We were able to include all of the files in the original HCRL Survival Analysis Dataset except the attack-free file for the Chevrolet Spark. We selected the Chevrolet Spark to serve as the “unknown” vehicle; therefore, we could not include its attack-free traffic in the training data. The unused file contains 136,935 samples. `hcr1-sa` lacks suppress and masquerade attacks; therefore, we were unable to create testing subsets `test_05_suppress` and `test_06_masquerade`.

In Table 7, we provide a numeric outline of the `hcr1-sa` dataset (compare to Table 8). The original HCRL Survival Analysis dataset contained a total of 446,626 attack-free samples: 192,517 for the Kia Soul, 117,174 for the Hyundai YF Sonata, and 136,935 for the Chevrolet Spark.

Rajapaksha et al. [103] and Verma et al. [146] point out that the attack-free data relevant to each vehicle (approximately 60 to 90 seconds per vehicle) is not sufficient to properly train a machine learning model. In addition, we found that, like `hcr1-ch`, `hcr1-sa` is formatted inconsistently. The CAN traces are stored in `.txt` files. The attack traces appear to be comma-separated, but two of the three attack-free files use a mix of commas and spaces as delimiters. Practitioners planning to write a script to preprocess the `hcr1-sa` should be aware that they will need to tweak the script to handle these inconsistent files.

4.2 The `can-train-and-test` and `can-train-and-test-v1.5` Datasets

The `can-train-and-test` dataset. Collected and curated by Lampe and Meng, researchers at the Technical University of Denmark, the `can-train-and-test` dataset [64, 67, 60, 68] contains attack-free CAN traffic as well as nine unique attacks:

1. Denial of Service
2. Fuzzing
3. Systematic
4. Gear Spoofing
5. RPM Spoofing
6. Speed Spoofing
7. Combined Spoofing
8. Standstill

9. Interval

The *denial of service* attack in `can-train-and-test` leveraged high-priority arbitration IDs to disrupt legitimate communications. Like `hcr1-ch` and `hcr1-sa`, some of the DoS traces in `can-train-and-test` were flooded with the highest-priority arbitration ID, “000.” Unlike `hcr1-ch` and `hcr1-sa`, many of the DoS traces in `can-train-and-test` were flooded with the highest-priority legitimate arbitration ID—that is, the highest-priority arbitration ID that occurred during attack-free traffic (e.g., “002” for the Subaru Forester). The *fuzzing* attack injected randomized CAN messages (random arbitration ID, random data field). The *systematic* attack sequentially or “systematically” injected every possible arbitration ID—valid or not.

The gear, RPM, and speed *spoofing* attacks targeted specific arbitration IDs to confuse and disrupt the components and functionalities associated with them. For example, in some vehicles, a gear spoofing attack will force the vehicle to shift into the spoofed gear. The *combined* spoofing attacks spoof two or three CAN signals (e.g., gear, RPMs, speed) simultaneously. The *standstill* attack, if successful, will put the vehicle in “standstill” mode, disabling the accelerator.

The *interval* attack employs a more sophisticated timing-based strategy. Rather than continuous message injection, it monitors for a specific target ID and immediately transmits contradictory values when legitimate messages appear. When a message with the target arbitration ID is transmitted, the attacker immediately transmits a contradictory message. This timing-based approach exploits CAN’s periodic messaging pattern: the spoofed message remains active until the next legitimate message appears, which is then immediately countered with another spoofed message.

This strategy creates an asymmetric impact: legitimate messages have minimal influence due to immediate contradiction, while spoofed messages remain effective for the full interval until the next legitimate message. For instance, if a legitimate message indicates “speed is zero,” the attacker immediately transmits “speed is 50,” causing the dashboard to display 50 until the next legitimate message arrives. By leveraging message timing, the attacker can maintain control while transmitting fewer messages, potentially reducing detectability.

Traffic was collected from four different vehicles—a Chevrolet Impala, a Chevrolet Traverse, a Chevrolet Silverado, and a Subaru Forester. Many attacks were conducted against a live vehicle traveling at speed; due to safety considerations, some attacks were simulated.

According to Lampe and Meng, the `can-train-and-test` dataset was designed to help researchers evaluate a proposed IDS under specific conditions (e.g., evaluate the IDS against a known vehicle and an unknown attack) [64,

67]. Within each sub-dataset (#1 - #4), there are two training subsets and four testing subsets, as follows:

- `train_01_attack_free`: Train the IDS (attack-free traffic)
- `train_02_with_attacks`: Train the IDS (attack-free and attack traffic)
- `test_01_known_vehicle_known_attack`: Evaluate the IDS against a known vehicle type and a known attack.
- `test_02_unknown_vehicle_known_attack`: Evaluate the IDS against an unknown vehicle type and a known attack.
- `test_03_known_vehicle_unknown_attack`: Evaluate the IDS against a known vehicle type and an unknown attack.
- `test_04_unknown_vehicle_unknown_attack`: Evaluate the IDS against an unknown vehicle type and an unknown attack.

The `can-train-and-test-v1.5` dataset. Kidmose and Meng later extended the `can-train-and-test` dataset into the `can-train-and-test-v1.5` dataset [54], which contains two additional attacks—suppress and masquerade—in its two additional testing subsets, as follows:

- `test_05_suppress`: Test the IDS against a suppress attack.
- `test_06_masquerade`: Test the IDS against a masquerade attack.

The *suppress* attack removes or “suppresses” CAN messages with a particular arbitration ID—that is, CAN messages transmitted by a particular ECU. This attack mimics an attacker who disrupts an ECU’s communications—or disables an ECU in its entirety. As such, during a suppress attack, the “suppressed” ECU’s arbitration ID disappears from the CAN bus.

For the suppress attack, Kidmose and Meng removed messages programmatically by scripting the removal of specific messages from an attack-free trace. While this method introduces longer inter-message intervals, it reflects nuanced real-world communication dynamics. CAN networks typically operate at an average busload of 50% [102], with utilization occasionally reaching 70% and brief bursts approaching 100% (the J1939 standard requires tolerance of 100% utilization for ten milliseconds [147, 40]).

However, there are some differences between programmatic message deletion and a genuine suppress attack:

- During actual message bursts (approaching 100% utilization), a programmatic approach may introduce unnatural gaps that would be dynamically filled in a real-world scenario. Similarly, bus errors, which necessitate message retransmission by ECUs, naturally increase

busload. These real-world dynamics would sometimes fill potential communication gaps, a nuance absent from simplified programmatic simulation approaches.

- In an actual suppress attack, the data and communication patterns of ECUs dependent on the suppressed messages would likely change, another nuance absent from simplified programmatic simulation approaches.

These nuanced differences suggest that while programmatic attack simulation provides valuable insights, it cannot fully reproduce the intricate communication dynamics of a genuine CAN bus suppress attack. The impact of these nuanced differences varies depending on the intrusion detection system. For example, IDSs that monitor standard arbitration identifiers (and raise an alert when they disappear) would be minimally affected by such variations.

To develop a more authentic suppress attack, researchers would need to actively suppress target messages before they reach the CAN bus or cause them to be disregarded as errors upon bus transmission, ideally before an ECU can act on them. Multiple sophisticated techniques exist for message suppression:

- **Diagnostic Attack.** Miller and Valasek [85, 84] achieved ECU control by exploiting the Unified Diagnostic Services (UDS) protocol, which requires the vehicle to be stationary or moving extremely slowly (0 to 10 miles per hour [82]). Using UDS, they could place an ECU in Bootrom mode and initiate reprogramming. Once in this mode, the ECU remains unresponsive even if vehicle speed increases, as it lacks valid firmware to execute normal operations. Lee et al. [72] employed a similar approach, subsequently transforming the suppress attack into a masquerade attack by injecting messages into the communication gaps left by the suppressed ECU.
- **Counter Value Exploitation.** This suppress attack exploits vehicle-specific counter values and error handling mechanisms. Miller and Valasek [85, 84] demonstrated counter value exploitation by targeting a parking assist module that expects to receive messages with specific counter values. By transmitting a carefully constructed message with the correct counter value immediately before the legitimate message’s expected arrival, they forced the parking assist module (the recipient ECU) to disregard the authentic transmission and reset its error flags.
- **Transmit Error Counter Manipulation.** Lee et al. [72] leveraged CAN bus error handling and confinement rules to create a bus-off attack. By intentionally generating transmission errors, they incrementally increased the ECU’s Transmit Error Counter (TEC). When the TEC exceeds 255, the ECU enters a bus-off state, effectively suppressing its communication. While this attack

is temporary—the ECU will eventually recover and return to normal operations—it provides a window of opportunity for life-threatening attacks.

These methods demonstrate the complex and nuanced approaches possible in authentic message suppression, highlighting the sophisticated techniques beyond simple programmatic message deletion.

The *masquerade* attack comes in two flavors: untimed or timed. In an *untimed* masquerade attack, the attacker disrupts or disables a legitimate ECU, eliminating confliction. Then, the attacker “masquerades” as the legitimate ECU. The attacker does not attempt to match the timing (e.g., inter-message interval) of the legitimate ECU. In contrast, in a *timed* masquerade attack, the attacker carefully matches the timing of the legitimate ECU. The attacker might transmit a spoofed message each time a legitimate message is intercepted.

Automotive networks utilize a hierarchical structure where a CAN-based backbone network connects to various lower-level networks (typically LIN) through gateway units [93]. A central gateway manages inter-network communication, translating protocols and routing messages between domains via processor-based software implementations [126].

A practical vector for message tampering—and timed masquerade attacks—involves compromising network gateways [90]. When messages traverse these gateways (e.g., sensor data moving from a LIN subnet to the CAN backbone [93]), an attacker with gateway access could intercept and modify message content while preserving timing characteristics. Since the modified messages maintain legitimate arbitration IDs and timing patterns, they may evade detection methods that rely on timing analysis. Multiple real-world examples demonstrate the feasibility of such attacks: Miller and Valasek [82, 83] successfully reprogrammed a 2014 Jeep Cherokee’s protective gateway, while Nie et al. [91, 51] achieved similar gateway compromise in a Tesla Model S, both enabling arbitrary CAN message injection. Attack feasibility depends primarily on the vehicle’s specific network architecture, gateway security measures, and potential remote access vectors.

In all, the `can-train-and-test-v1.5` dataset contains eleven unique attacks. The extended dataset, which includes the four original testing subsets as well as the suppress and masquerade subsets, is available in the `can-train-and-test-v1.5` repository [61].

Table 6 provides a high-level overview of the entire `can-train-and-test-v1.5` dataset, while Table 8 constitutes a deep dive into sub-dataset #1 (`set_01` in the repository). Table 6 is comparable to Tables 5 and 9, while Table 8 is comparable to Table 7. `can-train-and-test-v1.5` contains 10,962,946 attack-free samples: `set_01` contains 3,652,306; `set_02` contains 2,299,718; `set_03` contains 3,669,787; and `set_04` contains 1,341,135.

In [54] (Table 2), Kidmose and Meng analyzed `can-train-and-test` and reported the number of training samples, the number of testing samples, and the total number of samples for each sub-dataset. In this paper, we analyze `can-train-and-test-v1.5`, which includes two new testing subsets—`test_05_suppress` and `test_06_masquerade`. As such, our numbers differ from those reported in [54].

4.3 The UNIMORE Bus-Off, DAGA, and Ventus Datasets

UNIMORE Bus-Off dataset (`bus-off`). Captured and curated by the Web Engineering and Benchmarking Laboratory at the University of Modena and Reggio Emilia, the `bus-off` dataset [131, 132] contains attack-free data (referred to as “clean” data by the authors) as well as two types of suppress attacks, namely,

1. Inhibition
2. Shutdown

In an *inhibition* attack, the attacker completely and permanently disables the target ECU. Consequently, the target ECU’s arbitration ID disappears from the CAN bus. The target ECU’s arbitration ID does not appear in the inhibition attack traces. The *shutdown* attack is perhaps more subtle—the attack temporarily disrupts the target ECU’s communications (e.g., by temporarily disabling the target ECU). In the shutdown attack traces, the target ECU’s arbitration ID disappears and then reappears.

All attack-free data was collected from a real vehicle—a 2016 Volvo V40 Kinetic. The attack traces were generated by removing specific messages from the attack-free traces. Unfortunately, simulated attacks struggle to achieve the same fidelity as attacks conducted in a real vehicle under driving conditions. Though Stabili and Marchetti [131, 132] do not describe how, exactly, they removed the target CAN messages from the attack-free CAN traces, an analysis of the original attack-free traces and the simulated attack traces suggests that the target CAN messages were removed programmatically. We compared an attack trace, `ID130_V40_01.csv`, to the attack-free trace from which it was constructed, `V40_01.csv`. These traces come from our reformatted version of the `bus-off` dataset, found in the `can-unimore-curated-v3` repository. Although we reformatted several features, we did not change any values; importantly, we did not adjust the value of the timestamp in any way. Given that we are analyzing the value of the timestamp, which did not change, we can safely use the reformatted version of the `bus-off` dataset for this analysis.

We found that the timestamps were not adjusted to compensate for the gaps left by missing (i.e., removed) messages, mirroring the approach in `can-train-and-test-v1.5`. Listing 1 contains a snippet of attack-free traffic from

Listing 1 Comma-delimited attack-free traffic from a 2016 Volvo V40 Kinetic (UNIMORE Bus-Off dataset). The message that will be removed during the inhibition attack is shown in blue.

```
0.024412,3D0,810F6100661780C3,0
0.025279,290,0000000000000000,0
0.028052,0E0,627103E000806000,0
0.029553,288,F9DF30CF00000080,0
0.030968,0F5,00800080BF7F0100,0
0.03117,008,F1808C8E00800080,0
0.031175,120,008000807FFA5589,0
0.031452,130,FFFF1D00003FF11,0
0.03275,150,000FCF7F0F703C0,0
0.032756,010,008000002FB031F8,0
0.033046,1D0,0000000000000061,0
0.03305,291,0000000000000000,0
0.033054,400,020B011601010102,0
0.033401,405,46870031002F2F02,0
0.035556,125,E5587FE5C1D0D734,0
```

Listing 2 Comma-delimited inhibition attack traffic from a 2016 Volvo V40 Kinetic (UNIMORE Bus-Off dataset). This snippet corresponds to the attack-free snippet shown in Listing 1.

```
0.024412,3D0,810F6100661780C3,0
0.025279,290,0000000000000000,0
0.028052,0E0,627103E000806000,0
0.029553,288,F9DF30CF00000080,0
0.030968,0F5,00800080BF7F0100,0
0.03117,008,F1808C8E00800080,0
0.031175,120,008000807FFA5589,0
0.03275,150,000FCF7F0F703C0,0
0.032756,010,008000002FB031F8,0
0.033046,1D0,0000000000000061,0
0.03305,291,0000000000000000,0
0.033054,400,020B011601010102,0
0.033401,405,46870031002F2F02,0
0.035556,125,E5587FE5C1D0D734,0
```

the bus-off dataset (the message that will be removed during the inhibition attack is shown in blue). Listing 2 contains the corresponding snippet of inhibition attack traffic. We referred back to the original, unaltered UNIMORE Bus-Off dataset and confirmed our finding.

As discussed in Section 4.2, simple message deletion approximates real-world communication dynamics remarkably well. CAN networks' typical busload of 50% (possibly up to 70%) means the gaps introduced by message deletion often mirror natural inter-message intervals [147, 40]. However, subtle differences emerge during high-utilization scenarios. Message bursts approaching 100% bus utilization would likely see waiting messages dynamically filling potential gaps, a nuance absent from programmatic deletion. Moreover, ECUs relying on suppressed messages would exhibit altered communication patterns in a genuine attack—a

behavioral complexity not captured by post-hoc message removal.

These nuanced variations suggest that while programmatic suppress attack simulation provides valuable insights, it may not perfectly represent real-world attack scenarios. The performance of an intrusion detection system against a deletion-generated trace could potentially differ from its performance against an actual suppress attack on a live vehicle, particularly for detection strategies sensitive to message timing and interdependent ECU behaviors.

Table 9 numerically breaks down the can-unimore-curated-v3 repository, which contains the final curated versions of the three UNIMORE datasets, including the bus-off dataset. For numerical breakdowns of the original, unaltered datasets, see Table 10.

UNIMORE DAGA dataset (daga). Also constructed by the Web Engineering and Benchmarking Laboratory, the daga dataset [133, 134] provides attack-free data as well as six different attacks:

1. Denial of Service
2. Message ID Fuzzing (i.e., Arbitration ID Fuzzing)
3. Payload Fuzzing (i.e., Data Field Fuzzing)
4. Single ID Replay (i.e., Single Arbitration ID Replay)
5. Arbitrary Sequence Replay
6. Ordered Sequence Replay

The *denial of service* attacks in the daga dataset leverage the highest-priority legitimate arbitration ID (i.e., the smallest legitimate arbitration ID by value). These attacks are similar to some of those found in the can-train-and-test-v1.5. The *message ID fuzzing* attack randomly selects an *invalid* arbitration ID (an arbitration ID not seen in attack-free traffic) and pairs it with a randomly generated data field. This fuzzing attack differs from the fuzzing attacks in hcr1-ch, hcr1-sa, and can-train-and-test-v1.5 in that it specifically selects an arbitration ID that is *not* found in legitimate, attack-free traffic. In the *payload fuzzing* attack, a *valid* arbitration ID is randomly selected and paired with a randomly generated data field.

During the *single ID replay* attack, attackers capture a legitimate CAN message; later, they inject (i.e., “replay”) that single message once per second. During the *arbitrary sequence replay* attack, attackers capture a number of legitimate CAN messages; later, they inject those legitimate CAN messages—in no particular order. The arbitrary sequence of legitimate CAN messages is injected once per second. The length of the sequence ranges from two to ten messages. Lastly, during the *ordered sequence replay* attack, attackers capture a sequence of legitimate CAN messages; later, they inject that legitimate sequence. The messages are injected in order, and the sequence can range in length from two messages to ten messages.

Again, all attack-free data was collected from a real vehicle—a 2016 Volvo V40 Kinetic. The attack data was simulated on top of the attack-free data. Consequently, the *daga* dataset faces train-test interdependence issues—as well as potential fidelity issues. If we were to train an IDS using all seven of the attack-free traces available in the original, unaltered dataset, then, during the testing phase, the IDS would be pitted against the same attack-free traces it saw during training—just with added attack messages. In effect, the *daga* dataset was constructed such that train-test interdependence would be a major concern (for more information, see Sections 6.5 and 7.3).

That said, the attacks in the *daga* dataset were simulated with much higher fidelity than those in the *bus-off* dataset. As mentioned earlier in this section, with the *bus-off* dataset, the target messages were simply removed (presumably programmatically), failing to account for two critical real-world effects: potential ECU responses to missing critical information, and the probability that waiting messages would fill the artificial gaps during high bus utilization periods (i.e., message bursts).

DAGA’s simulation architecture, by contrast, replicated the physical characteristics of a CAN bus network. Stabili et al. [133, 134] implemented the CAN bus as a breadboard and connected a laptop computer, a Raspberry Pi 4 board, and an Arduino Mega to it. Connections to the simulated “CAN bus” were mediated by CAN transceivers and resistors—just as they would be in an actual vehicle. The Raspberry Pi replayed the clean traces while the Arduino injected attack messages. The laptop recorded the new attack traces. Stabili et al.’s attention to the low-level details should mitigate many of the fidelity issues that often arise from simulations.

For example, during a DoS attack, we would expect the flood of attack messages to disrupt and displace legitimate messages, changing the order of messages (and, importantly, arbitration IDs) in the trace. In an attack-free scenario, a legitimate ECU might transmit a low-priority message at a certain time. However, during our DoS scenario, the legitimate ECU is forced to wait for a high-priority attack message to finish transmitting. By the time the high-priority attack message is finished, a second legitimate ECU wants to transmit a high-priority message. Now, the first ECU must wait for the high-priority message to finish transmitting. This interaction will change the order of legitimate arbitration IDs; the patterns found in an attack-free trace will differ from those found in an attack trace. Indeed, when we look at some of the DoS attacks in the *daga* dataset, that is exactly what we find; message ordering changes as high-priority messages jump ahead of delayed low-priority messages.

Even the *daga* dataset’s high-fidelity simulations have limitations, though. Whole CAN frames can be reordered

(or even dropped) when replayed by the Raspberry Pi, transmitted by CAN transceivers, and confronted with malicious messages. However, because the messages are replayed, not generated by actual ECUs, they cannot change as the vehicle’s status changes. This phenomenon is missing from *daga*’s simulations.

Table 9 numerically breaks down the *can-unimore-curated-v3* repository, which contains the final curated versions of the three UNIMORE datasets, including the *daga* dataset. For numerical breakdowns of the original, unaltered datasets, see Table 10.

UNIMORE Ventus dataset (*ventus*). As with the previous two datasets, the *ventus* dataset [98, 97] was developed by the Web Engineering and Benchmarking Laboratory and includes attack-free data as well as two types of attacks:

1. Injection
2. Removal

The *injection* attack adds or “injects” CAN messages, while the *removal* attack erases or “removes” CAN messages. The injection attack constitutes a replay attack; legitimate messages are captured and “replayed” at rates of 1, 10, 25, 50, or 100 messages per second. For the removal attack, messages with the target ECU’s arbitration ID are removed from the entire attack trace. The removal attack is equivalent to the *bus-off* dataset’s inhibition attack.

As with the previous two datasets, all of the attack-free data was collected from a 2016 Volvo V40 Kinetic, while all of the attack data was simulated on top of the attack-free data via either injection or removal techniques. As with the *bus-off* and *daga* datasets, the *ventus* dataset is afflicted with train-test interdependence issues—and potential fidelity issues. Fortunately, Pollicino, Stabili, and Marchetti [98, 97] simulated the *injection* attacks using a high-fidelity setup similar to the one used for the *daga* dataset. Unfortunately, for the *removal* attacks, they used a low-fidelity setup similar to the one used for the *bus-off* dataset (and for *can-train-and-test-v1.5*’s suppress attack).

Table 9 numerically breaks down the *can-unimore-curated-v3* repository, which contains the final curated versions of the three UNIMORE datasets, including the *ventus* dataset. For numerical breakdowns of the original, unaltered datasets, see Table 10.

For all of the experiments discussed in this paper, we use the final versions of the UNIMORE Bus-Off, DAGA, and Ventus datasets; that is, the versions found in the *can-unimore-curated-v3* repository (see Table 9). The unaltered UNIMORE datasets each contain seven attack-free traces and 8,000,000+ attack-free samples—UNIMORE Bus-Off contains 8,743,772 attack-free samples, UNIMORE DAGA contains 8,743,772 attack-free samples, and UNIMORE Ventus contains 8,654,387 attack-

free samples. The final versions of the bus-off, daga, and ventus datasets are all subdivided into two sub-datasets—set_01 and set_02. Each sub-dataset contains three of the seven attack-free traces—more than enough to properly train a machine learning model.

4.4 How to Access the Datasets

All of the datasets used in our study are publicly available online. Below, we provide links to the original, unaltered datasets—as well as some notes about how to find them:

The HCRL Car-Hacking dataset:

- ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

The HCRL Survival Analysis dataset:

- ocslab.hksecurity.net/Datasets/survival-ids

The can-train-and-test dataset:

- bitbucket.org/brooke-lampe/can-train-and-test/

The can-train-and-test-v1.5 dataset:

- bitbucket.org/brooke-lampe/can-train-and-test-v1.5/

The UNIMORE Bus-Off dataset:

- weblab.ing.unimore.it/people/stabili/resources/
- Click on “IEEE Vehicular Technology Conference 2019 - Fall (VTC2019 Fall)”

The UNIMORE DAGA dataset:

- weblab.ing.unimore.it/people/stabili/resources/
- Click on “DAGA Dataset (preview)”

The UNIMORE Ventus dataset:

- weblab.ing.unimore.it/people/stabili/resources/tcps.shtml

We reformatted and curated several of the above datasets for the purposes of our study. In particular, we modified the two HCRL datasets and the three UNIMORE datasets to match the format, organization, and size of the can-train-and-test and can-train-and-test-v1.5 datasets.

The two HCRL datasets are available in the can-train-and-test-v1.5 repository. The links are provided below:

The HCRL Car-Hacking dataset (hcrl-ch)

- bitbucket.org/brooke-lampe/can-train-and-test-v1.5/src/master/hcrl-ch/

The HCRL Car-Hacking dataset (hcrl-sa)

- bitbucket.org/brooke-lampe/can-train-and-test-v1.5/src/master/hcrl-sa/

We developed three different versions of the three UNIMORE datasets, which are stored in the following repositories: can-unimore-curated-v1, can-unimore-curated-v2, and can-unimore-curated-v3. The can-unimore-curated-v1 repository contains the minimally curated and trimmed versions of the UNIMORE datasets, while can-unimore-curated-v3 repository contains the highly curated and trimmed versions. The links are provided below:

can-unimore-curated-v1

- <https://bitbucket.org/brooke-lampe/can-unimore-curated-v1/>

can-unimore-curated-v2

- <https://bitbucket.org/brooke-lampe/can-unimore-curated-v2/>

can-unimore-curated-v3

- <https://bitbucket.org/brooke-lampe/can-unimore-curated-v3/>

5 Comparison of Datasets

In this section, we compare and contrast our six datasets—in terms of size, train-test split, class imbalance, etc. We begin by calculating the total size of each dataset. Note that, when we calculated the total number of samples in each dataset for Tables 5, 7, 6, and 9, we excluded duplicate CAN traces (CAN traces that appear in both train_01_attack-free and train_02_with_attacks). However, when we calculated the total size of each dataset, we included these duplicate CAN traces, so that practitioners can see how much memory each dataset will consume when downloaded. When calculating memory consumption, we included all CAN traces, but we excluded any file that was not a CAN trace (e.g., git files, helper files, etc.). Practitioners do not need to download and store these files in order to use the dataset.

As mentioned in Section 4.4, the can-train-and-test-v1.5 repository contains the reformatted, curated versions of hcrl-ch and hcrl-sa. When calculating the size of can-train-and-test-v1.5, we exclude hcrl-ch and hcrl-sa. The sizes of hcrl-ch and hcrl-sa are given separately. Practitioners can choose to download can-train-and-test-v1.5 without hcrl-ch and hcrl-sa.

In [64] and [67], Lampe and Meng reported the size of the can-train-and-test dataset. Below, we report the size of the can-train-and-test-v1.5 dataset, which contains two new testing subsets—test_05_suppress and test_06_masquerade. The two new testing subsets are responsible for the size differences between can-train-and-test and can-train-and-test-v1.5.

Total size of each dataset:

- hcrl-ch	0.753 GB
- hcrl-sa	0.077 GB
- can-train-and-test-v1.5	9.7 GB
- can-unimore-raw	106.1 GB
- bus-off	21.6 GB
- daga	27.2 GB
- ventus	57.3 GB
- can-unimore-curated-v1	35 GB
- bus-off	6.7 GB
- daga	9.2 GB
- ventus	19.1 GB
- can-unimore-curated-v2	16.8 GB
- bus-off	3.3 GB
- daga	4.5 GB
- ventus	9.0 GB
- can-unimore-curated-v3	9.6 GB
- bus-off	3.0 GB

- daga	2.5 GB
- ventus	4.1 GB

5.1 hcrl-ch and hcrl-sa vs. can-train-and-test-v1.5

Tables 5 and 6 provide numeric breakdowns of hcrl-ch and can-train-and-test-v1.5, respectively. Tables 7 and 8 enumerate the sizes (in CAN frames) of hcrl-sa and sub-dataset #1 of can-train-and-test-v1.5. In terms of CAN traffic samples (i.e., CAN frames), we can see that can-train-and-test-v1.5 is about an order of magnitude larger than either of the HCRL datasets; accordingly, it is better equipped to showcase the capabilities of machine learning—especially deep learning—IDSs, which notoriously require a lot of training data.

Tables 6 and 8 also reveal that the can-train-and-test-v1.5 dataset contains a high volume of testing data relative to training data—and relative to the traditional 70/30 to 80/20 range of train-test splits [22]. In part, the relatively high volume of testing data arises from the four independent testing subsets contained in each sub-dataset. However, Table 8 calls attention to the fact that some testing subsets are nearly as large—or even larger—than the training subset. Given an abundance of data, it is beneficial to evaluate a would-be intrusion detection system against a high volume of data. In the real world, an IDS would be continuously challenged by an unending stream of data. A large testing subset is more realistic than a small testing subset but is still more practicable than a never-ending real-world test.

5.2 can-train-and-test-v1.5 vs. bus-off, daga, and ventus

Looking back at Tables 6 and 9, we can see that sub-dataset #1 (set_01) of can-train-and-test-v1.5 is slightly larger than set_01 of bus-off, somewhat larger than set_01 of daga, and slightly smaller than set_01 of ventus. For reference, set_01 of can-train-and-test-v1.5 contains 55,582,992 total samples, compared to 41,151,602 total samples for bus-off, 32,695,426 total samples for daga, and 55,802,180 total samples for ventus.

We can see that, in general, can-unimore-curated-v3 is smaller than can-train-and-test-v1.5; however, can-train-and-test-v1.5 and can-unimore-curated-v3 are much closer in size than can-train-and-test-v1.5 and can-unimore-curated-v2 (see Table 12)—our previous iteration of preprocessing, trimming, and curating the three UNIMORE datasets.

Even so, one could argue that can-unimore-curated-v3 is at a disadvantage compared to can-train-and-test-v1.5, since it contains fewer samples. However,

set_01 of the bus-off dataset contains 18,206,709 *training* samples compared to can-train-and-test-v1.5's mere 11,460,705. Similarly, set_01 of the daga dataset contains 14,947,920 training samples (again, compared to can-train-and-test-v1.5's mere 11,460,705), while set_01 of the ventus dataset contains a whopping 27,270,266 training samples.

In fact, set_01 of the ventus dataset contains more training samples than any of can-train-and-test-v1.5's sub-datasets. Moreover, set_01, set_03, and set_04 of can-train-and-test-v1.5 contain fewer training samples than any sub-dataset in can-unimore-curated-v3.

Thus, the number of training samples in can-unimore-curated-v3 (per sub-dataset) is comparable to—or greater than—the number of training samples in can-train-and-test-v1.5 (per sub-dataset). Therefore, an IDS trained and tested by can-unimore-curated-v3 will be at least as well trained (with respect to the quantity of training data), as an IDS trained and tested by can-train-and-test-v1.5.

5.3 Class Imbalance

Below, we provide the class imbalance (ratio of attack-free samples to attack samples) for each dataset in our study. Note that some testing subsets (e.g., test_05_suppress in can-train-and-test-v1.5 and test_01_inhibition in bus-off) contain only suppress attacks. Suppress attacks remove (or “suppress”) CAN messages; they do not inject CAN messages. As such, if a testing subset contains only suppress attacks, then it lacks labeled attack messages. We cannot calculate class imbalance without labeled attack messages; therefore, we write “N/A,” meaning “not applicable,” whenever a testing subset contains only suppress attacks (for more information on suppress attacks, see Sections 4.2 and 4.3).

- hcrl-ch	
- train_02_with_attacks	7:1
- test_01_DoS	3:1
- test_02_fuzzing	4:1
- test_03_gear_spoofing	4:1
- test_04_rpm_spoofing	4:1
- hcrl-sa	
- train_02_with_attacks	11:1
- test_01_known_vehicle_known_attack	4:1
- test_02_unknown_vehicle_known_attack	5:1
- test_03_known_vehicle_unknown_attack	12:1
- test_04_unknown_vehicle_unknown_attack	8:1
- can-train-and-test-v1.5 (sub-dataset #1)	
- train_02_with_attacks	213:1
- test_01_known_vehicle_known_attack	89:1
- test_02_unknown_vehicle_known_attack	38:1
- test_03_known_vehicle_unknown_attack	413:1
- test_04_unknown_vehicle_unknown_attack	927:1
- test_05_suppress	N/A
- test_06_masquerade	75:1

- bus-off (sub-dataset #1)	
- train_02_with_attacks	N/A
- test_01_inhibition	N/A
- test_02_shutdown	N/A
- daga (sub-dataset #1)	
- train_02_with_attacks	108:1
- test_01_DoS	17:1
- test_02_arbitration_id_fuzzing	1602:1
- test_03_data_field_fuzzing	1068:1
- test_04_single_arbitration_id_replay	457:1
- test_05_arbitrary_sequence_replay	71:1
- test_06_ordered_sequence_replay	66:1
- ventus (sub-dataset #1)	
- train_02_with_attacks	61:1
- test_01_injection	43:1
- test_02_removal	N/A

As outlined above, `can-train-and-test-v1.5` demonstrates extreme class imbalance—the ratio of attack-free samples to attack samples can be as high as 927:1. Extreme class imbalance is associated with a number of challenges, particularly when it comes to capturing performance metrics. However, extreme class imbalance is also realistic to the problem at hand; a million messages will cross the CAN bus within about five to ten minutes [63]. Since vehicles routinely travel the world’s roads for months, years, and even decades without incident, we can see that class imbalance in the real world is even steeper. Therefore, `can-train-and-test-v1.5` enables researchers to assess the capabilities of a proposed IDS in a more authentic scenario. If the proposed IDS has a problematic false positive rate, it will almost certainly be revealed when the ratio of attack-free samples to attack samples is 927:1. The `daga` dataset demonstrates even greater class imbalance—one testing subset has an attack-free:attack ratio of 1602:1, and another has a ratio of 1068:1. The extreme class imbalance of the `daga` dataset means that it is even more qualified to expose problematic false positives.

6 Methodology

Our investigation centered around the `can-train-and-test-v1.5` dataset [64, 67, 61]. We extracted features from the dataset and evaluated the impact of said features on the performance of machine learning IDSs. Next, we pitted `can-train-and-test-v1.5` against two well-established open-access intrusion detection datasets, namely, the HCRL Car-Hacking dataset [123, 130, 122] and the HCRL Survival Analysis dataset [33, 34]. Then, we compared the `can-train-and-test-v1.5` to three newer datasets, specifically, UNIMORE Bus-Off [131, 132], DAGA [133, 134], and Ventus [98, 97]. These three datasets are more similar to `can-train-and-test-v1.5` in terms of data volume and class imbalance. Finally, we used the three UNIMORE datasets to investigate the impact of train-test interdependence on the performance of our machine learning models.

For all of our experiments, we leveraged the following sixteen machine learning models:

1. Traditional machine learning, *supervised*
 - (a) Gaussian Naive Bayes (GNB)
 - (b) *K*-Nearest Neighbors (KNN)
 - (c) Linear Regression (Lin Reg)
 - (d) Logistic Regression (Log Reg)
 - (e) Linear Support Vector Machine (Lin SVM)
 - (f) Decision Tree (DT)
 - (g) Extra Trees (ET)
 - (h) Gradient Boosting (GB)
 - (i) Random Forest (RF)
2. Traditional machine learning, *unsupervised*
 - (a) Isolation Forest (IF)
 - (b) K-Means (KM)
 - (c) Mini-Batch K-Means (MBKM)
 - (d) Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)
 - (e) Local Outlier Factor (LOF)
3. Deep learning, *supervised*
 - (a) Multi-Layer Perceptron (MLP)
4. Deep learning, *unsupervised*
 - (a) Bernoulli Restricted Boltzmann Machine (BRBM)

As shown above, there are fourteen traditional machine learning models and two deep learning models. At the same time, there are ten supervised models and six unsupervised models. Our machine learning models were constructed using the `scikit-learn` [96] Python library.

6.1 Preliminaries

In order to compare the `can-train-and-test-v1.5` dataset to five others—HCRL Car Hacking, HCRL Survival Analysis, UNIMORE Bus-Off, UNIMORE DAGA, and UNIMORE Ventus—we first had to standardize them. Our five comparison datasets were developed by a number of different authors at two different universities; as such, they are organized and formatted differently. Some datasets—e.g., the HCRL Car Hacking dataset, the HCRL Survival Analysis dataset—are not even internally consistent, formatting-wise (see Section 4.1). Therefore, we wrote preprocessing scripts in Python—with the help of the `pandas` [94] Python library—to reformat our five comparison datasets to match `can-train-and-test-v1.5`.

One important preprocessing step was to ensure that the arbitration ID was stored as a 3-digit hexadecimal value. According to the CAN specification, the arbitration ID must be eleven bits, which corresponds to a 3-digit hexadecimal value between “000” and “7FF” (the maximum allowed by the 11-bit format). To elaborate, the minimum 11-bit value

in binary is “0000000000” (or “000” in hex) and the maximum is “1111111111” (or “7FF” in hex). When communicating the minimum binary value, all eleven zeroes must be transmitted for arbitration purposes (i.e., prioritization)—one cannot simply transmit “0”. [80, 136]. Therefore, when we express the arbitration ID in hexadecimal, we use exactly three digits.

During the preprocessing phase, if the arbitration was shorter than three digits, we padded with leading zeroes to meet the length requirement. If it was longer, we removed an excess leading zero. However, when we reformatted the raw datasets, we did not change the length of the data field in any way—that is, we did not add or remove leading or trailing zeroes. The CAN specification explicitly allows for variable-length data fields; therefore, we preserved the variable lengths of the data fields when constructing our preprocessed versions of the raw datasets. We did encounter CAN traces in which the data field had been subdivided (using commas or spaces) into separate bytes. Since the data field is not subdivided in `can-train-and-test-v1.5`, we removed the delimiters and consolidated the data field into one feature.

During our feature evaluation (see Section 6.2), our Python-based driver program was responsible for subdividing—or not subdividing—the data field before the input data was passed along to the IDS. For all of our experiments, we had to equalize the lengths of the data fields before passing them along to our machine learning models. To that end, we prepended zeroes until the data field was eight bytes—or sixteen hex digits—long.

We represented the `arbitration ID` and `data field` features as Python `float` variables, maintaining this approach whether the data field was subdivided or not. While integers could have been equally appropriate, `float` data types provided convenient conversion methods. Our use case involved fixed-size numbers well within Python’s floating-point precision limits [92].

We explicitly avoided `byte` or `bytearray` types due to compatibility constraints with machine learning libraries like `scikit-learn`. The `float` representation ensured seamless integration with our computational tools and machine learning models, facilitating straightforward feature engineering and model training.

It is important to note that we captured time in the form of a `timestamp` feature, *not* a `time delta` feature. In `can-train-and-test-v1.5`, each CAN trace begins with the following timestamp: `2023-01-01 00:00:00.000000000` (January 1, 2023, at midnight). Subsequent timestamps are offset appropriately so the `time delta` (the time difference between two consecutive timestamps) is unchanged. The timestamps in the HCRL and UNIMORE datasets were reformatted to match `can-train-and-test-v1.5`; however, the actual timestamp values

were not altered. The value of a given timestamp in the preprocessed version of the dataset is the same as its value in the original, unaltered dataset. An in-depth evaluation of the `time delta` feature is beyond the scope of this paper (see instead [53]).

Due to significant class imbalance across the datasets—in some instances, exhibiting extreme disparities—we adopted a nuanced approach to performance metrics. Utilizing `scikit-learn`’s capabilities, we implemented the “weighted” option for precision, recall, and F1-score [96]. The weighted approach addresses the challenges posed by class imbalance, providing a more robust evaluation metric that accounts for the uneven distribution of classes. This approach calculates metrics for each label and determines the average weighted by support (the number of true instances for each label) [119, 120, 118]. The “weighted” option is unavailable for accuracy [117]. As `scikit-learn` does not provide predefined g-mean and false positive rate (FPR) metrics, we computed these additional metrics ourselves.

6.2 Feature Evaluation

For the `can-train-and-test-v1.5` feature evaluation, we selected two features: (1) the `timestamp` feature and (2) the `data field` feature.

With the `timestamp` feature, we sought to evaluate its inherent value. Hossain and Islam [39] emphasize that—in the context of machine learning for intrusion detection—it is crucial to distinguish between informative and irrelevant features. They caution that overfitting—in which a complex model fails to generalize to new data—frequently occurs when noisy or redundant features are included in the input data. If we determined that the `timestamp` feature was uninformative, then we could exclude it from future experiments. However, if the `timestamp` proved valuable, then its absence could lead to underfitting; in other words, if the `timestamp` was explaining some of the variation in the data, then our machine learning models might struggle to capture patterns in the data and might underfit [78].

To assess the value of the `timestamp` feature, we conducted one series of experiments *including* the `timestamp`, followed by another series *excluding* it. Our goal was to answer question such as:

1. Is the `timestamp` feature relevant? Irrelevant? Redundant?
2. If the `timestamp` contains both information and noise, then does it contain more information or more noise?
3. Does including the `timestamp` contribute to overfitting?
4. Does excluding the `timestamp` contribute to underfitting?

The results are shown in Tables 14 and 15, respectively.

With the `data field` feature, we sought to evaluate the impact of subdividing the `data field` into eight data bytes—each data byte treated as a separate feature. If subdividing the `data field` did not improve performance, then it was a needless preprocessing step. As such, we conducted one set of experiments in which the `data field` was subdivided into eight distinct `data field byte` features, followed by one set of experiments in which the `data field` remained intact as a single feature. The results are shown in Tables 14 and 16, respectively.

Input Features. For the results shown in Table 14, the machine learning models were provided with the following ten features:

1. Timestamp
2. Arbitration ID
3. Byte #1
4. Byte #2
5. Byte #3
6. Byte #4
7. Byte #5
8. Byte #6
9. Byte #7
10. Byte #8

For Table 15, the `timestamp` feature was excluded, leading to the following list of nine features:

1. Arbitration ID
2. Byte #1
3. Byte #2
4. Byte #3
5. Byte #4
6. Byte #5
7. Byte #6
8. Byte #7
9. Byte #8

For Table 16, the `data field` feature was not subdivided, leading to the following list of just three features:

1. Timestamp
2. Arbitration ID
3. Data field

In all cases, the `timestamp` feature was represented by a Python `datetime` variable, while the `arbitration ID` feature and `data field` feature (or individual `data field byte` features) were represented by Python `float` variables.

6.3 `hcr1-ch` and `hcr1-sa` vs. `can-train-and-test-v1.5`

To compare `can-train-and-test-v1.5` to `hcr1-ch` and `hcr1-sa`, we opted to evaluate all sixteen of our machine

learning models against the three datasets, then analyze the results. To leverage our machine learning models against `hcr1-ch` and `hcr1-sa`, we had to conduct a few preprocessing steps (see also Section 6.1). Essentially, we reformatted `hcr1-ch` and `hcr1-sa` as comma-separated values (CSV) files containing only the expected features in the expected format. For `hcr1-ch`, we removed the leading zero from the `arbitration ID` feature and added a leading zero to the first byte of the `data field` feature. Then, we concatenated the eight comma- or space-delimited bytes of the `data field` into a single feature (our Python-based driver program is responsible for subdividing—or *not* subdividing—the `data field`). Lastly, we removed the `data length code` (DLC) feature, and we converted the R/T flag to a 0/1 flag to match `can-train-and-test-v1.5`. For `hcr1-sa`, we followed the same preprocessing steps, and then we changed the file extension from `.txt` to `.csv`.

The `can-train-and-test-v1.5` dataset was curated to facilitate scenario-specific evaluation. As mentioned in Section 4.2, one scenario available in `can-train-and-test-v1.5` is the “known vehicle, unknown attack” pairing. In this scenario, the vehicle has been seen in training (known vehicle), but the attack has not been seen in training (unknown attack). Because `can-train-and-test-v1.5` was designed to fill this specialized role, it is not directly comparable to `hcr1-ch` and `hcr1-sa`. As such, we opted to conduct two different comparisons: (1) a conventional comparison to a conventionally-partitioned dataset, and a scenario-based comparison to a dataset that partitioned according to scenarios, similar to `can-train-and-test-v1.5` itself. The data in the `hcr1-ch` dataset comes from exactly one vehicle, so it is not suited to our scenario-based comparison. However, three different vehicles—a Chevrolet Spark, a Hyundai YF Sonata, and a Kia Soul—are represented in the `hcr1-sa` dataset. In addition, `hcr1-sa` contains three different attacks: DoS, fuzzing, and malfunction. Thus, `hcr1-sa` can be configured as a scenario-based dataset—akin to `can-train-and-test-v1.5`. Since `hcr1-sa` provides relatively little data on a per vehicle basis—especially attack-free data (see Section 4.1)—we are curious about the possibility of underfitting [78].

Below, we describe how `hcr1-ch` and `hcr1-sa` were organized and partitioned with respect to training and testing data:

- For `hcr1-ch`, we divide each attack into traditional train-test splits (~75% for training, ~25% for testing). All of the attack-free data is used as training data. We use the train splits for training and the test splits for testing.
- For `hcr1-sa`, we select two of the three vehicles and two of the three attacks for training. These vehicles and attacks will be the *known* vehicles and *known* attacks. We withhold one vehicle and one attack; these will be

the *unknown* vehicle and *unknown* attack. All of the data in the Survival Analysis dataset was included in the scenario-based version of the dataset—except the attack-free file for the *unknown* vehicle. In keeping with the `can-train-and-test-v1.5`, we include the attack-free files in the training data only. Since the *unknown* vehicle cannot be seen in training, its attack-free file was not used. The unused file contains 136,935 samples.

6.4 `can-train-and-test-v1.5` vs. `bus-off`, `daga`, and `ventus`

With the UNIMORE Bus-Off, DAGA, and Ventus datasets, our intention, for each dataset, was create one attack-free training subset (`train_01_attack_free`), one training subset with attacks (`train_02_with_attacks`), and a number of testing subsets—as appropriate to the number and types of attacks available in the given dataset. We planned to reuse the strategies that went into the `hcr1-ch` dataset in order to construct the curated `bus-off`, `daga`, and `ventus` datasets. For example, we planned to use all of the available data for either training or testing, we planned to use a $\sim 75/\sim 25$ train-test split, and we did *not* plan to construct sub-datasets.

However, we quickly realized that the original, unaltered UNIMORE datasets were simply too large—far too large. The raw, full-size datasets (collectively referred to as `can-unimore-raw`; see Table 10) consumed a whopping 106.1 GB of memory: the raw `bus-off` dataset was 21.6 GB in size, the raw `daga` dataset was even larger at 27.2 GB, and the raw `ventus` dataset was by far the largest at 57.3 GB. Our machine learning models—which run on CPUs—would be prohibitively slow if forced to process 57.3 GB of data. Some of them would simply run out of memory resources and terminate with an error message. For reference, `can-train-and-test-v1.5`'s footprint is 9.7 GB—spread across four sub-datasets—and `hcr1-ch` consumes a mere 0.753 GB of memory.

We realized we would need to trim and curate the raw datasets. Our first iteration of trimmed, curated datasets was `can-unimore-curated-v1` (see Table 11); we quickly determined that it was still too large. We further trimmed and curated the datasets, culminating in `can-unimore-curated-v2` (see Table 12). In `can-unimore-curated-v2`, the `bus-off`, `daga`, and `ventus` datasets are each sub-divided into two sub-datasets (`set_01` and `set_02`). While significantly trimmed, `can-unimore-curated-v2` preserved all of the unique attack types found in the three UNIMORE datasets. We conducted a complete set of experiments with `can-unimore-curated-v2`—the results are available in the supplementary material (see “`data-extended.xlsx`,” an Excel spreadsheet).

When we reviewed the results of our `can-unimore-curated-v2` experiments, we realized that a number of

our machine learning models were still running out of either time or memory. For the `bus-off` dataset, 3 out of 32 experiments ran out of either time or memory; for the `daga` dataset, it was 14 out of 32; and for the `ventus` dataset, it was 21 out of 32. Therefore, we trimmed and curated the three UNIMORE datasets one more time, developing the `can-unimore-curated-v3` (see Tables 9 and 13). The `can-unimore-curated-v3` repository, containing the final trimmed and curated versions of the three UNIMORE datasets, is well-suited to practitioners who have little or no access to dedicated machine learning hardware and software. Many machine learning models—including deep learning models such as the Multi-Layer Perceptron—can be trained and tested on `can-unimore-curated-v3` in a timely fashion (less than seven days) without a GPU. Across all three UNIMORE datasets, we conducted 96 experiments, and only 2 of those 96 experiments ran out of either time or memory.

As discussed in Section 5.2, `can-unimore-curated-v3`'s `bus-off`, `daga`, and `ventus` datasets are slightly smaller—with respect to the total number of samples—than `can-train-and-test-v1.5`. However, the three UNIMORE datasets contain more training samples—per sub-dataset—than all but one of the sub-datasets in `can-train-and-test-v1.5`. And the volume of training data in a dataset is much more important than the volume of testing data—hence the common 70/30 and 80/20 train-test splits [22, 39, 78]. With respect to training data quantity, an IDS trained and tested by `can-unimore-curated-v3` will be at least as well trained—on average—as one trained and tested with `can-train-and-test-v1.5`.

See Table 10, Table 11, Table 12, and Table 13 for `can-unimore-raw` (original and unaltered), `can-unimore-curated-v1` (minimally trimmed and curated), `can-unimore-curated-v2` (moderately trimmed and curated), and `can-unimore-curated-v3` (highly trimmed and curated), respectively.

6.5 Train-Test Interdependence

The UNIMORE Bus-Off, DAGA, and Ventus datasets collectively contain eight attacks (some of which overlap). All eight attacks were simulated—some with greater fidelity than others. Importantly, both the high- and low-fidelity simulations were conducted on top of attack-free traces; that is, the authors took an attack-free trace and either injected or removed messages. As discussed in Section 4.3, the suppress (or “removal”) attacks in the `bus-off` and `ventus` datasets were probably conducted programmatically; the target messages are removed, but the remainder of the trace is not altered in any way. This simulation strategy is relatively low fidelity; the `timestamp` feature is left with gaps where the “removed” messages once were, even during message bursts

Table 10: can-unimore-raw¹

Dataset	# of samples in attack-free (clean) captures	# of samples in attack (infected) captures	Total # of samples
bus-off	8,743,772	180,339,296	189,083,068
daga	8,743,772	247,557,238	256,301,010
ventus	8,654,387	531,003,538	539,657,925
TOTAL	26,141,931	958,900,072	985,042,003

¹ can-unimore-raw denotes the original, raw Bus-Off, DAGA, and Ventus datasets constructed by Stabili and Marchetti [131]; Stabili et al. [133]; and Pollicino, Stabili, and Marchetti [98], respectively. The raw datasets do not contain sub-datasets.

Table 11: can-unimore-curated-v1^{1,2}

Dataset	# of training samples	# of testing samples	Total # of samples
bus-off	141,886,413	47,196,655	189,083,068
daga	191,494,702	64,806,308	256,301,010
ventus	401,530,248	138,127,677	539,657,925
TOTAL	734,911,363	250,130,640	985,042,003

¹ can-unimore-curated-v1 denotes the minimally preprocessed Bus-Off, DAGA, and Ventus datasets. We reformatted and reorganized the datasets to accommodate our machine learning models. These datasets do not contain sub-datasets.

² We exclude samples in duplicate CAN traces.

Table 12: can-unimore-curated-v2^{1,2}

Dataset	# of training samples	# of testing samples	Total # of samples
bus-off	43,254,585	47,196,655	90,451,240
daga	56,837,897	64,806,308	121,644,205
ventus	114,450,604	138,127,677	252,578,281
TOTAL	214,543,086	250,130,640	464,673,726

¹ can-unimore-curated-v2 denotes our first attempt to significantly resize and reorganize the Bus-Off, DAGA, and Ventus datasets to minimize errors (e.g., “out of memory”) during machine learning. Each of these datasets is split into two sub-datasets (not shown).

² We exclude samples in duplicate CAN traces.

Table 13: can-unimore-curated-v3^{1,2}

Dataset	# of training samples	# of testing samples	Total # of samples
bus-off	35,558,504	44,907,642	80,466,146
daga	30,373,648	34,734,339	65,107,987
ventus	53,719,367	56,727,490	110,446,857
TOTAL	119,651,519	136,369,471	256,020,990

¹ can-unimore-curated-v3 denotes our second and final attempt to significantly resize and reorganize the Bus-Off, DAGA, and Ventus datasets to minimize errors (e.g., “out of memory”) during machine learning. Each of these datasets is split into two sub-datasets (not shown).

² We exclude samples in duplicate CAN traces.

(approaching 100% busload), where gaps would likely be filled by waiting messages. By contrast, the daga dataset (as well as the ventus dataset’s “injection” attack) constitutes a high-fidelity simulation; both a combination of hardware and software was used to account for the low-level particulars of the CAN bus—down to the transceivers and resistors (see Section 4.3). Given its higher fidelity, we focus on the daga dataset in our analysis of train-test interdependence.

To analyze train-test interdependence in the daga dataset, we needed one sub-dataset that demonstrated train-

test interdependence, and one sub-dataset that did not (our baseline). Ultimately, we subdivided all three UNINMORE datasets into two sub-datasets. In sub-dataset #1 (set_01), the training data contained a specific attack-free trace, and the testing data contained the attack traces simulated on top of that specific attack-free trace. In sub-dataset #2 (set_02), however, none of the attack-free traces in the training data were related to the attack traces in the testing data. Thus, set_01 was curated to exhibit train-test interdependence, while set_02 (our baseline) was not.

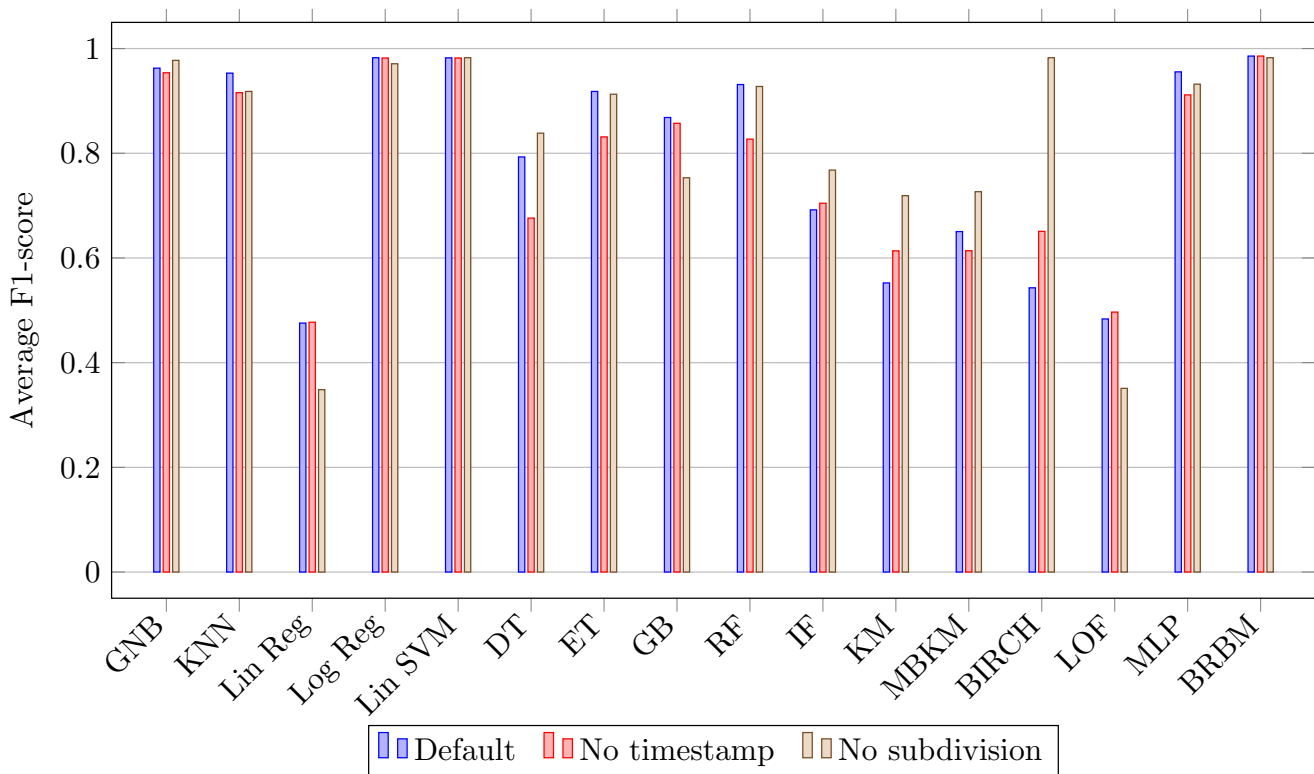


Fig. 1: Average F1-scores for can-train-and-test-v1.5 under the following conditions: (1) *with* timestamp and *with* data field subdivision (default), (2) *without* timestamp (with data field subdivision), and (3) *without* data field subdivision (with timestamp). This graph corresponds to Tables 14, 15, and 16, for the blue, red, and tan bars, respectively.

With this setup, we sought to answer the question “Can an IDS memorize an attack-free trace during training and use that information to identify attacks in related attack traces?” Until the start of the first attack, the attack traces are generally identical to their attack-free trace (i.e., the attack-free trace they originated from). Moreover, many attack traces are identical (or nearly identical) to their attack-free trace between attacks as well. Since Stabili et al. [133, 134] initialized the daga traces to 0.0, the timestamps are often identical as well. As such, we expected the IDS to perform much better against set_01 than set_02.

As mentioned in Section 6.4, we ran our full battery of experiments on can-unimore-curated-v2 (see “data-extended.xlsx,” an Excel spreadsheet, in the supplementary material). However, when we examined the data, we found that many of our experiments had run out of either time or memory. Upon examination of the daga dataset, we found that nine of our sixteen machine learning models completed both the set_01 and set_02 experiments. Of those, six had better F1-scores when pitted against set_01 than when pitted against set_02. For the three machine learning models that performed worse against set_01 (the opposite of our expectation), one was essentially equal: the difference was 0.0001. We could see that, for the experiments that ac-

tually ran to completion, set_01 had a significant advantage over set_02. The results of these experiments convinced us to develop can-unimore-curated-v3. We wanted to evaluate the impact of train-test interdependence on IDS performance when all of the daga experiments terminated without error. For our analysis of train-test interdependence in can-unimore-curated-v3, see Section 7.4.

The HCRL and can-train-and-test-v1.5 datasets differ fundamentally from the UNIMORE datasets in terms of data generation methodology, precluding analysis of potential machine learning advantages from training data reuse in attack data. In the HCRL datasets, attack traces were captured from real attacks on real vehicles, not constructed by modifying attack-free traces. Similarly, can-train-and-test-v1.5’s attack traces were either captured from live vehicle attacks or simulated using attack-free traces that were never published as training data.

In contrast, all of the attacks in the UNIMORE datasets were generated by modifying a handful of published attack-free traces. Moreover, the UNIMORE datasets employ a systematic attack generation approach: each dataset contains seven attack-free traces, and every attack trace is derived from one of these base traces. This one-to-many relationship between attack-free and attack traces—where each

attack-free trace generates multiple attack variants—enables us to investigate whether intrusion detection systems gain performance advantages from exposure to the underlying attack-free patterns during training. Theoretically, such exposure could give them an advantage when they see the same attack-free traces, plus attacks, during testing. Because the HCRL and `can-train-and-test-v1.5` datasets lack these “base traces,” we cannot use them for our train-test interdependence analysis.

7 Discussion of Results

In this section, we discuss the results of our feature evaluation, our HCRL vs. `can-train-and-test-v1.5` comparison, our `can-train-and-test-v1.5` vs. UNIMORE comparison, and our train-test interdependence experiments. In our discussion of results, we often summarize or aggregate our results, as we do not have space to cover every experiment individually. Therefore, additional results are available in the supplementary material (see “`data-extended.xlsx`,” an Excel spreadsheet).

7.1 Feature Evaluation

Tables 14, 15, and 16 present different configurations of experiments run on the `can-train-and-test-v1.5` dataset. Table 14 shows results using default parameters, where the `data field` is subdivided and the `timestamp` feature is included. Table 15 presents results without the `timestamp` feature, while Table 16 shows results without `data field` subdivision. For each configuration, the experiments are averaged on a per-model basis. The three experimental configurations are visualized in Figure 1, which compares their average F1-scores under the following conditions: (1) *with* timestamp and *with* data field subdivision (default), (2) *without* timestamp (with data field subdivision), and (3) *without* data field subdivision (with timestamp).

Reviewing Tables 14, 15, and 16, we observe that feature configuration impacts varied significantly by model. As shown in Figure 1, the Multi-Layer Perceptron (MLP) achieved its highest F1-score (0.9555) with both the timestamp and the data field subdivision, declining to 0.9115 without the timestamp and 0.9320 without the data field subdivision. The Gaussian Naive Bayes (GNB) model exhibited different behavior, with F1-scores of 0.9626 (with timestamp), 0.9538 (without timestamp), and 0.9775 (without subdivision)—performing best without data field subdivision. Linear Regression showed limited effectiveness across all configurations, with F1-scores of 0.4756, 0.4772, and 0.3484—*with* the timestamp and subdivision, *without* the timestamp, and *without* the subdivision, respectively—performing marginally better without the timestamp feature.

In summary, for the MLP model, the highest average F1-score was achieved *with* the timestamp *and* the subdivision. For GNB, the highest average F1-score was achieved *without* the subdivision. For the Linear Regression model, the highest average F1-score was achieved *without* the timestamp.

When it came to subdividing the `data field` feature, the results were too mixed to generalize; however, when it came to including—or excluding—the `timestamp` feature, the results were clearer cut. Anecdotally, we can see that excluding the timestamp had a catastrophic impact on the performance of the Random Forest model but enhanced the performance of the Isolation Forest model. In general, though, excluding the `timestamp` feature is detrimental. Many of the high-performing machine learning models—e.g., Gaussian Naive Bayes, Logistic Regression, and Linear Support Vector Machine—exhibited slight drops in performance when the timestamp was removed. With the timestamp, the F1-scores were 0.9626, 0.9825, and 0.9823, respectively; without the timestamp, the F1-scores were 0.9538, 0.9820, and 0.9820, respectively. The *k*-Nearest Neighbors model saw a more significant drop in performance—from 0.9531 to 0.9158, as did the Multi-Layer Perceptron—0.9555 to 0.9115. Many of the models that *improved* without the timestamp were medium- or poor-performing models—the BIRCH model’s performance increased from 0.5430 to 0.6508; the Local Outlier Factor model’s performance increased from 0.4835 to 0.4966.

A pattern emerges when we examine moderate- and high-performing models (F1-score > 0.75) under comparable conditions. When analyzing Tables 14 and 15, with the `data field` subdivision held constant, we can see that removing the timestamp *never* improved the average F1-score. Figure 1 enables us to visually confirm our findings by examining its bar graph representation. We focus on the blue bars (*with* timestamp) and the red bars (*without* timestamp). Our cut-off for moderate- to high-performing models is 0.75, but for ease of visual inspection, we will begin by examining blue or red bars which cross the 0.8 mark. We can see that the blue bars meet or exceed the red bars in all cases—GNB, KNN, Log Reg, Lin SVM, ET, GB, RF, MLP, RBM. Some pairs of bars appear equal in height—e.g., Log Reg, Lin SVM—but a detailed inspection of Tables 14 and 15 reveals marginally better performance *with* the timestamp. In the case of the BRBM model, the bars are actually equal—the F1-score is 0.9857, with or without the timestamp. The 0.75 mark is more difficult to pinpoint, but we can see that the Decision Trees model meets our criteria for moderate- to high-performing, and it also matches our pattern: there is a significant drop from “with timestamp” to “without timestamp.” With the Isolation Forest (IF) model, the pattern is reversed; the IF model performs better “without timestamp” than “with timestamp.” This model, how-

Table 14: can-train-and-test-v1.5: Averages *with* timestamp and *with* data field subdivision.¹

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9498	0.9787	0.9498	0.9626	0.9550	0.0397
<i>K</i> -Nearest Neighbors	0.9320	0.9830	0.9320	0.9531	0.9361	0.0597
Linear Regression	0.3360	0.9810	0.3360	0.4756	0.3330	0.6700
Logistic Regression	0.9856	0.9827	0.9856	0.9825	0.9908	0.0039
Linear Support Vector Machine	0.9866	0.9785	0.9866	0.9823	0.9921	0.0023
Decision Tree	0.7027	0.9853	0.7027	0.7930	0.7033	0.2962
Extra Trees	0.8783	0.9837	0.8783	0.9181	0.8819	0.1144
Gradient Boosting	0.7986	0.9839	0.7986	0.8683	0.8006	0.1974
Random Forest	0.8936	0.9842	0.8936	0.9313	0.8971	0.0993
Isolation Forest	0.5448	0.9766	0.5448	0.6919	0.5457	0.4534
K-Means	0.4118	0.9796	0.4118	0.5523	0.4092	0.5933
Mini-Batch K-Means	0.5186	0.9778	0.5186	0.6504	0.5195	0.4796
BIRCH	0.4823	0.9783	0.4823	0.5430	0.4831	0.5160
Local Outlier Factor	0.3621	0.9722	0.3621	0.4835	0.3609	0.6403
Multi-Layer Perceptron	0.9343	0.9839	0.9343	0.9555	0.9383	0.0576
Bernoulli Restricted Boltzmann Machine	0.9904	0.9811	0.9904	0.9857	0.9952	0.0000

¹ We exclude `test_05_suppress`, as it lacks labeled CAN frames and cannot be effectively evaluated.

Table 15: can-train-and-test-v1.5: Averages *without* timestamp (with data field subdivision).¹

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9346	0.9809	0.9346	0.9538	0.9397	0.0552
<i>K</i> -Nearest Neighbors	0.8753	0.9862	0.8753	0.9158	0.8780	0.1194
Linear Regression	0.3387	0.9809	0.3387	0.4772	0.3357	0.6673
Logistic Regression	0.9849	0.9835	0.9849	0.9820	0.9901	0.0046
Linear Support Vector Machine	0.9861	0.9784	0.9861	0.9820	0.9916	0.0028
Decision Tree	0.5864	0.9844	0.5864	0.6762	0.5857	0.4149
Extra Trees	0.7785	0.9856	0.7785	0.8313	0.7804	0.2176
Gradient Boosting	0.7814	0.9837	0.7814	0.8573	0.7833	0.2148
Random Forest	0.7613	0.9859	0.7613	0.8270	0.7627	0.2358
Isolation Forest	0.5598	0.9771	0.5598	0.7046	0.5607	0.4384
K-Means	0.4750	0.9795	0.4750	0.6136	0.4749	0.5252
Mini-Batch K-Means	0.4789	0.9796	0.4789	0.6139	0.4789	0.5211
BIRCH	0.5678	0.9807	0.5678	0.6508	0.5675	0.4328
Local Outlier Factor	0.3830	0.9700	0.3830	0.4966	0.3826	0.6176
Multi-Layer Perceptron	0.8647	0.9846	0.8647	0.9115	0.8675	0.1297
Bernoulli Restricted Boltzmann Machine	0.9904	0.9811	0.9904	0.9857	0.9952	0.0000

¹ We exclude `test_05_suppress`, as it lacks labeled CAN frames and cannot be effectively evaluated.

Table 16: can-train-and-test-v1.5: Averages *without* data field subdivision (with timestamp).¹

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9782	0.9798	0.9782	0.9775	0.9840	0.0102
<i>K</i> -Nearest Neighbors	0.8769	0.9804	0.8769	0.9181	0.8813	0.1143
Linear Regression	0.2416	0.9828	0.2416	0.3484	0.2372	0.7669
Logistic Regression	0.9666	0.9794	0.9666	0.9710	0.9717	0.0232
Linear Support Vector Machine	0.9883	0.9768	0.9883	0.9825	0.9941	0.0000
Decision Tree	0.7634	0.9843	0.7634	0.8384	0.7653	0.2328
Extra Trees	0.8687	0.9811	0.8687	0.9128	0.8728	0.1230
Gradient Boosting	0.6590	0.9844	0.6590	0.7532	0.6596	0.3397
Random Forest	0.8920	0.9808	0.8920	0.9276	0.8964	0.0991
Isolation Forest	0.6396	0.9780	0.6396	0.7679	0.6408	0.3579
K-Means	0.6028	0.9747	0.6028	0.7190	0.6060	0.3908
Mini-Batch K-Means	0.6124	0.9773	0.6124	0.7266	0.6139	0.3845
BIRCH	0.9883	0.9768	0.9883	0.9825	0.9941	0.0000
Local Outlier Factor	0.2813	0.9818	0.2813	0.3509	0.2784	0.7242
Multi-Layer Perceptron	0.8970	0.9805	0.8970	0.9320	0.9011	0.0947
Bernoulli Restricted Boltzmann Machine	0.9883	0.9768	0.9883	0.9825	0.9941	0.0000

¹ We exclude `test_05_suppress`, as it lacks labeled CAN frames and cannot be effectively evaluated.

Table 17: hcrl-ch, hcrl-sa: Performance metrics for the KNN model.

Dataset, testing subset	Accuracy	Precision	Recall	F1-score	G-mean	FPR
CH, #1	0.8419	0.8553	0.8419	0.8102	0.9142	0.0073
CH, #2	0.9580	0.9592	0.9580	0.9561	0.9775	0.0026
CH, #3	0.9088	0.9114	0.9088	0.8973	0.9500	0.0069
CH, #4	0.9336	0.9348	0.9336	0.9282	0.9630	0.0066
SA, #1	0.8944	0.8908	0.8944	0.8831	0.9365	0.0195
SA, #2	0.6836	0.7355	0.6836	0.7072	0.7255	0.2301
SA, #3	0.9393	0.9322	0.9393	0.9343	0.9593	0.0203
SA, #4	0.6662	0.7809	0.6662	0.7190	0.7026	0.2590

Table 18: can-train-and-test-v1.5: Performance metrics for the KNN model.

Sub-dataset, testing subset	Accuracy	Precision	Recall	F1-score	G-mean	FPR
#1, #1	0.9855	0.9850	0.9855	0.9853	0.9893	0.0069
#1, #2	0.9714	0.9497	0.9714	0.9604	0.9840	0.0033
#1, #3	0.9976	0.9964	0.9976	0.9965	0.9988	0.0000
#1, #4	0.9982	0.9979	0.9982	0.9981	0.9987	0.0008
#1, #5	0.9915	1.0000	0.9915	0.9957	0.9915	0.0085
#1, #6	0.9851	0.9872	0.9851	0.9860	0.9878	0.0094
#2, #1	0.9988	0.9988	0.9988	0.9988	0.9990	0.0007
#2, #2	0.7930	0.9966	0.7930	0.8824	0.7932	0.2065
#2, #3	0.9261	0.9951	0.9261	0.9594	0.9272	0.0717
#2, #4	0.7658	0.9955	0.7658	0.8637	0.7656	0.2345
#2, #5	0.9432	1.0000	0.9432	0.9708	0.9432	0.0568
#2, #6	0.9559	0.9885	0.9559	0.9715	0.9583	0.0392
#3, #1	0.9912	0.9913	0.9912	0.9899	0.9956	0.0000
#3, #2	0.9461	0.9643	0.9461	0.9546	0.9537	0.0387
#3, #3	0.9980	0.9963	0.9980	0.9971	0.9990	0.0000
#3, #4	0.9489	0.9580	0.9489	0.9534	0.9589	0.0310
#3, #5	0.9958	1.0000	0.9958	0.9979	0.9958	0.0042
#3, #6	0.9857	0.9810	0.9857	0.9833	0.9904	0.0048
#4, #1	0.9825	0.9822	0.9825	0.9768	0.9911	0.0002
#4, #2	0.7357	0.9947	0.7357	0.8448	0.7361	0.2635
#4, #3	0.9820	0.9811	0.9820	0.9766	0.9908	0.0004
#4, #4	0.7117	0.9543	0.7117	0.8096	0.7156	0.2805
#4, #5	0.9999	1.0000	0.9999	1.0000	0.9999	0.0001
#6, #6	0.9808	0.9653	0.9808	0.9730	0.9895	0.0017

Table 19: can-unimore-curated-v3: Performance metrics for the KNN model.

Dataset	Sub-dataset, testing subset	Accuracy	Precision	Recall	F1-score	G-mean	FPR
bus-off	#1, #1	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
bus-off	#1, #2	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
bus-off	#2, #1	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
bus-off	#2, #2	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
daga	#1, #1	0.9950	0.9954	0.9950	0.9951	0.9948	0.0053
daga	#1, #2	0.9941	0.9988	0.9941	0.9964	0.9944	0.0053
daga	#1, #3	0.9938	0.9981	0.9938	0.9959	0.9942	0.0053
daga	#1, #4	0.9928	0.9961	0.9928	0.9944	0.9937	0.0053
daga	#1, #5	0.9811	0.9729	0.9811	0.9770	0.9879	0.0053
daga	#1, #6	0.9803	0.9713	0.9803	0.9758	0.9875	0.0053
daga	#2, #1	0.9769	0.9756	0.9769	0.9754	0.9859	0.0051
daga	#2, #2	0.9943	0.9988	0.9943	0.9965	0.9946	0.0051
daga	#2, #3	0.9930	0.9961	0.9930	0.9945	0.9939	0.0051
daga	#2, #4	0.9940	0.9981	0.9940	0.9960	0.9944	0.0051
daga	#2, #5	0.9807	0.9720	0.9807	0.9762	0.9878	0.0051
daga	#2, #6	0.9814	0.9729	0.9814	0.9771	0.9881	0.0051
ventus	#1, #1	0.9765	0.9551	0.9765	0.9657	0.9878	0.0008
ventus	#1, #2	0.9993	1.0000	0.9993	0.9996	0.9993	0.0007
ventus	#2, #1	0.9575	0.9547	0.9575	0.9561	0.9686	0.0202
ventus	#2, #2	0.9815	1.0000	0.9815	0.9907	0.9815	0.0185

Table 20: hcr1-sa, can-train-and-test-v1.5 (set_01): Performance metrics for the MLP model.

Dataset, sub-dataset, testing subset	Accuracy	Precision	Recall	F1-score	G-mean	FPR
SA, #1	0.9810	0.9819	0.9810	0.9812	0.9811	0.0187
SA, #2	0.7407	0.7517	0.7407	0.7461	0.7875	0.1627
SA, #3	0.9708	0.9712	0.9708	0.9678	0.9848	0.0009
SA, #4	0.7736	0.7954	0.7736	0.7844	0.8158	0.1396
CT&T, #1, #1	0.9873	0.9860	0.9873	0.9866	0.9910	0.0053
CT&T, #1, #2	0.9694	0.9539	0.9694	0.9609	0.9815	0.0062
CT&T, #1, #3	0.9976	0.9955	0.9976	0.9965	0.9988	0.0000
CT&T, #1, #4	0.9983	0.9979	0.9983	0.9981	0.9988	0.0007

¹ We exclude test_05_suppress and test_06_masquerade; no equivalent testing subsets are available in hcr1-sa.

ever, does not meet our cut-off, the average F1-scores are 0.6919 (with timestamp) and 0.7046 (without timestamp).

When we turn our attention to the question of data field subdivision, with the timestamp held constant, we see some intriguing behavior. Most models demonstrate consistent relative performance across different experimental configurations—that is, strong performers tend to remain strong, and weaker models tend to stay weaker, with only minor variations. The Logistic Regression, Linear Support Vector Machine, and Bernoulli Restricted Boltzmann Machine exemplify this consistency particularly well. However, the BIRCH model stands as a remarkable exception to this trend. However, the BIRCH model stands as a remarkable exception to this trend. With data field subdivision, it achieves a very modest F1-score of 0.5430, yet without subdivision, its performance soars to 0.9825—the most dramatic performance swing observed in our analysis. This stark contrast suggests that BIRCH’s clustering algorithm may interact with data field organization in a fundamentally different way than other models.

7.2 hcr1-ch and hcr1-sa vs. can-train-and-test-v1.5

The results of the experiments run against the HCRL Car Hacking dataset can be found in Table 21, while the results for the HCRL Survival Analysis dataset are presented in Table 22. To provide a broader perspective, we aggregate these two HCRL datasets in Table 30 in the appendix, allowing us to compare the can-train-and-test-v1.5 dataset against a generalized view of HCRL’s performance (see Section B). Figure 2 visualizes the data—specifically, the average F1-scores—in the form of a bar graph. The blue, red, and tan bars correspond to the HCRL Car Hacking dataset (hcr1-ch), the HCRL Survival Analysis dataset (hcr1-sa), and the can-train-and-test-v1.5 dataset, respectively.

Zooming in on our comparison, we examine Table 21 and Table 23, in which hcr1-ch is juxtaposed against can-train-and-test-v1.5 without testing subsets #5 and #6 (test_05_suppress and test_06_masquerade, respectively). Similarly, to compare hcr1-sa to can-train-and-

test-v1.5, we refer to Tables 22 and Table 23. Both Table 14 (referenced in our earlier discussion) and Table 23 contain the average metrics of the can-train-and-test-v1.5 dataset with default parameters; however, Table 14 includes testing subset #6 (masquerade) in its averages, whereas Table 23 does not. The hcr1-ch and hcr1-sa datasets do not contain CAN traffic data equivalent to testing subset #6; accordingly, in Table 23, we exclude testing subset #6 from the averages to enhance our perspective and our comparison.

When we average our experiments, we can see that many machine learning models perform very well (F1-score > 0.9) against the can-train-and-test-v1.5 dataset. A few models perform moderately well (0.9 > F1-score > 0.7), and a few perform poorly (F1-score ~ 0.5). Looking at the average model performance for the hcr1-ch dataset, we see that performance is generally lower: one model performs very well (F1-score > 0.9), many models perform well (0.9 > F1-score > 0.7), a few models perform poorly (F1-score ~ 0.5), and one model performs very poorly (F1-score < 0.4). Similarly, when averaging model performance for the hcr1-sa dataset, we see one model that performs very well (F1-score > 0.9), many models that perform moderately well (0.9 > F1-score > 0.7), a few models that perform poorly (F1-score ~ 0.5), and a few models that perform very poorly (F1-score < 0.4). The one model that performed very well against the hcr1-ch dataset was also the one model that performed well against the hcr1-sa dataset—the Gradient Boosting model. Interestingly, when pitted against can-train-and-test-v1.5, it achieved an average F1-score of 0.8897, below our cut-off for the “performed very well” classification.

A model-by-model analysis of Figure 2 reveals that eleven models performed best when pitted against can-train-and-test-v1.5, while five performed best against hcr1-ch. None of the models performed best against hcr1-sa—the red bar was never the tallest of the three bars. The systematic underperformance of our sixteen machine learning models when tested against hcr1-sa likely stems from the limited number of samples in the dataset, which may

Table 21: hcr1-ch ONLY: Averages with default parameters.

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.8082	0.7137	0.8082	0.7303	0.8956	0.0073
K-Nearest Neighbors	0.9106	0.9152	0.9106	0.8980	0.9512	0.0059
Linear Regression	0.4587	0.8381	0.4587	0.4870	0.3963	0.6575
Logistic Regression	0.8147	0.7663	0.8147	0.7354	0.9022	0.0005
Linear Support Vector Machine	0.8129	0.7590	0.8129	0.7308	0.9014	0.0002
Decision Tree	0.7143	0.8253	0.7143	0.7339	0.7238	0.2638
Extra Trees	0.8928	0.8860	0.8928	0.8809	0.9249	0.0405
Gradient Boosting	0.9646	0.9701	0.9646	0.9648	0.9684	0.0276
Random Forest	0.8604	0.9034	0.8604	0.8713	0.8602	0.1395
Isolation Forest	0.4273	0.7554	0.4273	0.4680	0.3900	0.6415
K-Means	0.7824	0.8046	0.7824	0.7834	0.8031	0.1730
Mini-Batch K-Means	0.7908	0.8117	0.7908	0.7911	0.8098	0.1680
BIRCH	0.8060	0.7179	0.8060	0.7580	0.8751	0.0485
Local Outlier Factor	0.1752	0.4945	0.1752	0.0594	0.0163	0.9980
Multi-Layer Perceptron	0.8892	0.9037	0.8892	0.8662	0.9428	0.0000
Bernoulli Restricted Boltzmann Machine	0.8114	0.6589	0.8114	0.7271	0.9007	0.0000

Table 22: hcr1-sa ONLY: Averages with default parameters.

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.7739	0.7954	0.7739	0.7839	0.8152	0.1411
K-Nearest Neighbors	0.7959	0.8349	0.7959	0.8109	0.8310	0.1322
Linear Regression	0.2749	0.8452	0.2749	0.2836	0.2213	0.8152
Logistic Regression	0.8440	0.7627	0.8440	0.8003	0.9028	0.0337
Linear Support Vector Machine	0.8550	0.7628	0.8550	0.8053	0.9149	0.0205
Decision Tree	0.6654	0.8758	0.6654	0.7232	0.6561	0.3520
Extra Trees	0.8129	0.8397	0.8129	0.8177	0.8530	0.1047
Gradient Boosting	0.9107	0.9011	0.9107	0.9031	0.9315	0.0467
Random Forest	0.8521	0.8833	0.8521	0.8597	0.8710	0.1090
Isolation Forest	0.4815	0.8013	0.4815	0.5590	0.4718	0.5369
K-Means	0.4396	0.7764	0.4396	0.5207	0.4284	0.5820
Mini-Batch K-Means	0.7287	0.7851	0.7287	0.7511	0.7635	0.2000
BIRCH	0.3274	0.8022	0.3274	0.3830	0.2926	0.7384
Local Outlier Factor	0.1959	0.7267	0.1959	0.1804	0.1382	0.9023
Multi-Layer Perceptron	0.8665	0.8751	0.8665	0.8699	0.8923	0.0805
Bernoulli Restricted Boltzmann Machine	0.8724	0.7628	0.8724	0.8135	0.9337	0.0000

Table 23: can-train-and-test-v1.5: Averages with default parameters; *exclude* testing subsets #5 and #6.¹

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9495	0.9789	0.9495	0.9627	0.9548	0.0398
K-Nearest Neighbors	0.9208	0.9836	0.9208	0.9467	0.9248	0.0712
Linear Regression	0.3283	0.9797	0.3283	0.4710	0.3254	0.6774
Logistic Regression	0.9856	0.9830	0.9856	0.9822	0.9909	0.0036
Linear Support Vector Machine	0.9861	0.9779	0.9861	0.9818	0.9917	0.0025
Decision Tree	0.7154	0.9861	0.7154	0.7979	0.7159	0.2835
Extra Trees	0.8667	0.9846	0.8667	0.9102	0.8700	0.1266
Gradient Boosting	0.8270	0.9844	0.8270	0.8897	0.8291	0.1687
Random Forest	0.8986	0.9842	0.8986	0.9347	0.9022	0.0941
Isolation Forest	0.5363	0.9783	0.5363	0.6842	0.5366	0.4630
K-Means	0.4093	0.9787	0.4093	0.5484	0.4066	0.5959
Mini-Batch K-Means	0.5180	0.9774	0.5180	0.6484	0.5189	0.4802
BIRCH	0.4854	0.9779	0.4854	0.5476	0.4861	0.5132
Local Outlier Factor	0.3907	0.9699	0.3907	0.5119	0.3898	0.6110
Multi-Layer Perceptron	0.9355	0.9844	0.9355	0.9561	0.9397	0.0562
Bernoulli Restricted Boltzmann Machine	0.9905	0.9812	0.9905	0.9858	0.9952	0.0000

¹ This evaluation was conducted to compare the can-train-and-test-v1.5 dataset with the HCRL Car Hacking (hcr1-ch) and Survival Analysis (hcr1-sa) datasets. Testing subsets #5 and #6 were *not* averaged into the performance metrics in this table, since there is no equivalent data in either of the HCRL datasets.

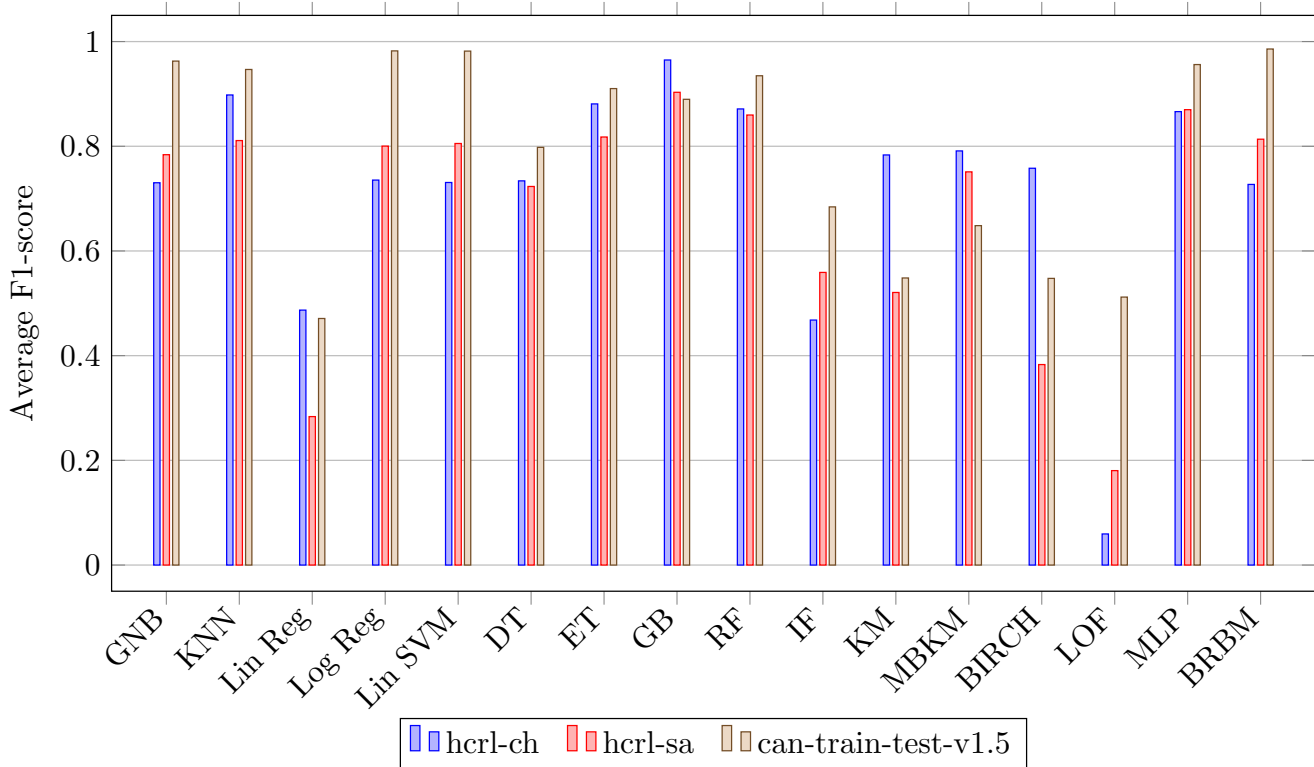


Fig. 2: Average F1-scores for hcr1-ch, hcr1-sa, and can-train-and-test-v1.5. This graph corresponds to Tables 21, 22, and 23, for the blue, red, and tan bars, respectively.

have inhibited the models’ ability to learn robust patterns during training.

In Tables 17 (hcr1-ch and hcr1-sa) and 18 (can-train-and-test-v1.5), we provide all of the metrics of the k -Nearest Neighbors (KNN) model—all datasets, all sub-datasets, and all testing subsets. Referring back to Tables 21, 22, and 23, we know that the KNN model performed significantly better when pitted against can-train-and-test-v1.5 than when pitted against either hcr1-ch or hcr1-sa. Tables 17 and 18 confirm that the KNN model’s performance is significantly higher for can-train-and-test-v1.5 than for hcr1-ch and hcr1-sa; the F1-scores for can-train-and-test-v1.5 are generally in the upper 0.8s to the upper 0.9s, while the F1-scores for hcr1-ch are in the mid-0.8s to mid-0.9s and the F1-scores for hcr1-sa are in the lower 0.7s to lower 0.9s.

Lastly, in Table 20 we assess the performance of the Multi-Layer Perceptron model when pitted against the can-train-and-test-v1.5 dataset and the hcr1-sa dataset. We conduct experiments that showcase the MLP model’s capabilities against four testing subsets: test_01_known_vehicle_known_attack, test_02_unknown_vehicle_known_attack, test_03_known_vehicle_unknown_attack, and test_04_unknown_vehicle_unknown_attack. For testing subsets #1 and #3,

the F1-score are fairly close: 0.9866 and 0.9965 for can-train-and-test-v1.5; 0.9812 and 0.9678 for hcr1-sa. However, when we look at testing subsets #2 and #4, the MLP model performs significantly better when pitted against the can-train-and-test-v1.5 dataset. The F1-scores are 0.9609 and 0.9981 for can-train-and-test-v1.5, whereas, for hcr1-sa, the F1-scores are 0.7461 and 0.7844. In all testing subsets, the MLP performed better against can-train-and-test-v1.5 than against hcr1-sa.

Since testing subsets #2 and #4 both concern *unknown* vehicles, we speculate that hcr1-sa lacked sufficient training data, leading the MLP to *underfit* to the training data [78]. hcr1-sa’s limited quantity of training data—especially attack-free training data—seems to support this interpretation (see Section 4.1).

However, can-train-and-test-v1.5’s sub-dataset #1 uses a Chevrolet Impala as the known vehicle and a Chevrolet Silverado as the unknown vehicle. The Silverado is a newer vehicle, and its CAN traffic contains arbitration IDs associated with features that the Impala does not have. However, they are both Chevrolet vehicles, and the arbitration IDs—and even some data fields—overlap substantially. In comparison, hcr1-sa uses the Hyundai YF Sonata and the Kia Soul as known vehicles, while the Chevrolet Spark

is chosen as the unknown vehicle. Since Hyundai is Kia’s parent company, the Sonata and Spark traces (the known vehicle traces) are substantially similar, whereas the Spark is very different.

To see how substantial overlap between the *known* vehicle’s data and the *unknown* vehicle’s data impacts the performance of the Multi-Layer Perceptron, we look at sub-datasets #2, #3, and #4 of `can-train-and-test-v1.5` (see “`data-extended.xlsx`,” an Excel spreadsheet, in the supplementary material). First, we examine the results of the experiments on sub-dataset #2, where the known vehicle was a Chevrolet Traverse and the unknown vehicle was a Subaru Forester—completely different vehicles from completely different manufacturers with completely different CAN traffic. For testing subsets #1 and #3, which concern *known* vehicles, the F1-scores are 0.9988 and 0.8904 (the latter is somewhat low compared to sub-dataset #1). For testing subsets #2 and #4, which concern *unknown* vehicles, the F1-scores are 0.8774 and 0.8512 (both are somewhat low compared to sub-dataset #1).

Sub-datasets #3 and #4 also contain vastly different known and unknown vehicles. For sub-dataset #3, the known and unknown vehicles are the Chevrolet Silverado and the Subaru Forester, respectively, while for sub-dataset #4, the known vehicle is the Subaru Forester and the unknown vehicle is the Chevrolet Traverse. In spite of the major differences between the known and unknown vehicles, sub-dataset #3’s F1-scores are similar to sub-dataset #1’s. In fact, for the known vehicle testing subsets, the F1-scores were 0.9928 and 0.9971; for the unknown vehicle testing subsets, the F1-scores were 0.9653 and 0.9580. F1-scores for the unknown vehicle were marginally lower, but still in the mid-0.9s. Sub-dataset #4 is much the same—testing subsets #1 and #3 yielded F1-scores of 0.9787 and 0.9761, respectively; testing subsets #2 and #4 gave F1-scores of 0.9509 and 0.9193, respectively. We can see that, in general, the MLP model performs slightly better when pitted against a known vehicle than when pitted against an unknown vehicle; however, the performance difference is slim compared to `hcr1-sa`. Whereas `hcr1-sa`’s dropped from the 0.9s to the 0.7s, `can-train-and-test-v1.5`’s performance only dropped from the 0.9s to the 0.8s in the worst case (sub-dataset #2). In most cases (sub-datasets #1, #3, and #4), it only dropped from the upper 0.9s to the lower 0.9s. In light of this information, we rule out a similarity advantage in `can-train-and-test-v1.5`, leading us to conclude that the MLP likely underfitted to `hcr1-sa` as a result of limited training data.

7.3 `can-train-and-test-v1.5` vs. `bus-off`, `daga`, and `ventus`

As highlighted in Section 1, this paper is an extension of *can-sleuth: Investigating and Evaluating Automotive Intrusion Detection Datasets*, originally presented at the 2024 European Interdisciplinary Cybersecurity Conference [54]. Therefore, when we added the three UNIMORE datasets to our evaluation, we leveraged the same experimental setup, machine learning models, evaluation strategies, and metrics as [54]. Unfortunately, our original experiments were ill-equipped to gauge an IDS’s performance against suppress attacks; thus, our new experiments are similarly ill-equipped. The UNIMORE Bus-Off dataset contains only suppress attacks; accordingly, its experimental results are uninformative.

The UNIMORE Ventus dataset contains two types of attacks—*injection* and *removal* (i.e., *suppress*). For our evaluation, we focus on the `ventus` dataset’s *injection* attacks and we exclude the uninformative *removal* attacks.

Though the results are uninformative, we include them as a benchmark, so practitioners can see what happens when an evaluation that depends on labeled data faces unlabelable suppress attacks. Our UNIMORE Bus-Off benchmark demonstrates what will happen if practitioners attempt to treat a suppress attack dataset as a labeled dataset. Our benchmark also serves as a warning: if practitioners treat the entire UNIMORE Ventus dataset as a labeled dataset—even though it contains suppress attacks—they will be introducing the uninformative metrics we see in Table 24 into their data.

Tables 24, 25, and 26 contain the results of the experiments run against the three UNIMORE datasets (`bus-off`, `daga`, and `ventus`, respectively). The experiments are averaged on a per-model basis. For comparison with `can-train-and-test-v1.5`, refer to Table 23 and Figure 3.

Figure 3 visualizes the average F1-scores across all four datasets in the form of a bar graph, with color-coding as follows: blue for UNIMORE Bus-Off (`bus-off`), red for UNIMORE DAGA (`daga`), tan for UNIMORE Ventus (`ventus`), and black for `can-train-and-test-v1.5`.

7.3.1 `can-train-and-test-v1.5` vs. `bus-off`

Testing subset #5 (`test_05_suppress`) of `can-train-and-test-v1.5` contains suppress attacks. As shown in Table 18, the *k*-Nearest Neighbors model achieved perfect precision (1.0000) for all four sub-datasets. The KNN also achieved F1-scores of 0.99+ for three of the four sub-datasets. As shown in Table 19, when pitted against the `bus-off` dataset, the KNN model achieved perfect results for all metrics—perfect 1.0000s for accuracy, precision, recall, F1-

Table 24: can-unimore-curated-v3, bus-off: Averages with default parameters.¹

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
K-Nearest Neighbors	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
Linear Regression	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
Logistic Regression	Error	Error	Error	Error	Error	Error
Linear Support Vector Machine	Error	Error	Error	Error	Error	Error
Decision Tree	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
Extra Trees	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
Gradient Boosting	Error	Error	Error	Error	Error	Error
Random Forest	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
Isolation Forest	0.3069	1.0000	0.3069	0.4696	0.3069	0.6931
K-Means	0.4357	1.0000	0.4357	0.6070	0.4357	0.5643
Mini-Batch K-Means	0.5024	1.0000	0.5024	0.6665	0.5024	0.4976
BIRCH	0.7981	1.0000	0.7981	0.8865	0.7981	0.2019
Local Outlier Factor	0.6076	1.0000	0.6076	0.7554	0.6076	0.3924
Multi-Layer Perceptron	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
Bernoulli Restricted Boltzmann Machine	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000

¹ The UNIMORE Bus-Off dataset contains only suppress attacks, which lack labeled CAN frames and cannot be effectively evaluated. The table shows baseline IDS performance, not detection capability.

Table 25: can-unimore-curated-v3, daga: Averages with default parameters.

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9902	0.9806	0.9902	0.9853	0.9950	0.0000
K-Nearest Neighbors	0.9881	0.9872	0.9881	0.9875	0.9914	0.0052
Linear Regression	0.2228	0.9818	0.2228	0.3456	0.2175	0.7875
Logistic Regression	0.9856	0.9718	0.9856	0.9786	0.9927	0.0000
Linear Support Vector Machine	0.9856	0.9718	0.9856	0.9786	0.9927	0.0000
Decision Tree	0.9908	0.9891	0.9908	0.9899	0.9934	0.0041
Extra Trees	0.9908	0.9890	0.9908	0.9899	0.9934	0.0040
Gradient Boosting	0.9943	0.9898	0.9943	0.9920	0.9968	0.0006
Random Forest	0.9918	0.9891	0.9918	0.9904	0.9944	0.0030
Isolation Forest	0.2920	0.9815	0.2920	0.4356	0.2875	0.7169
K-Means	0.5590	0.9658	0.5590	0.7062	0.5619	0.4352
Mini-Batch K-Means	0.4975	0.9730	0.4975	0.6503	0.4976	0.5023
BIRCH	0.8584	0.9704	0.8584	0.9100	0.8644	0.1296
Local Outlier Factor	0.5915	0.9822	0.5915	0.7222	0.5903	0.4109
Multi-Layer Perceptron	0.9930	0.9883	0.9930	0.9904	0.9963	0.0004
Bernoulli Restricted Boltzmann Machine	0.9856	0.9718	0.9856	0.9786	0.9927	0.0000

Table 26: can-unimore-curated-v3, ventus: Averages with default parameters; *exclude* testing subset #2.¹

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9401	0.9543	0.9401	0.9469	0.9509	0.0381
K-Nearest Neighbors	0.9670	0.9549	0.9670	0.9609	0.9782	0.0105
Linear Regression	0.3831	0.9729	0.3831	0.5278	0.3767	0.6295
Logistic Regression	0.3831	0.9729	0.3831	0.5278	0.3767	0.6295
Linear Support Vector Machine	0.9778	0.9638	0.9778	0.9677	0.9887	0.0003
Decision Tree	0.9683	0.9549	0.9683	0.9616	0.9794	0.0092
Extra Trees	0.9658	0.9549	0.9658	0.9603	0.9770	0.0117
Gradient Boosting	0.9768	0.9551	0.9768	0.9658	0.9880	0.0005
Random Forest	0.9684	0.9549	0.9684	0.9616	0.9795	0.0091
Isolation Forest	0.2986	0.9499	0.2986	0.4386	0.2948	0.7089
K-Means	0.4973	0.9502	0.4973	0.6437	0.4983	0.5006
Mini-Batch K-Means	0.4793	0.9484	0.4793	0.6260	0.4801	0.5190
BIRCH	0.6040	0.9699	0.6040	0.6888	0.6033	0.3972
Local Outlier Factor	0.5217	0.9639	0.5217	0.6639	0.5202	0.4812
Multi-Layer Perceptron	0.9733	0.9551	0.9733	0.9641	0.9845	0.0041
Bernoulli Restricted Boltzmann Machine	Out of memory	Out of memory	Out of memory	Out of memory	Out of memory	Out of memory

¹ We exclude `test_02_removal`, as it lacks labeled CAN frames and cannot be effectively evaluated.

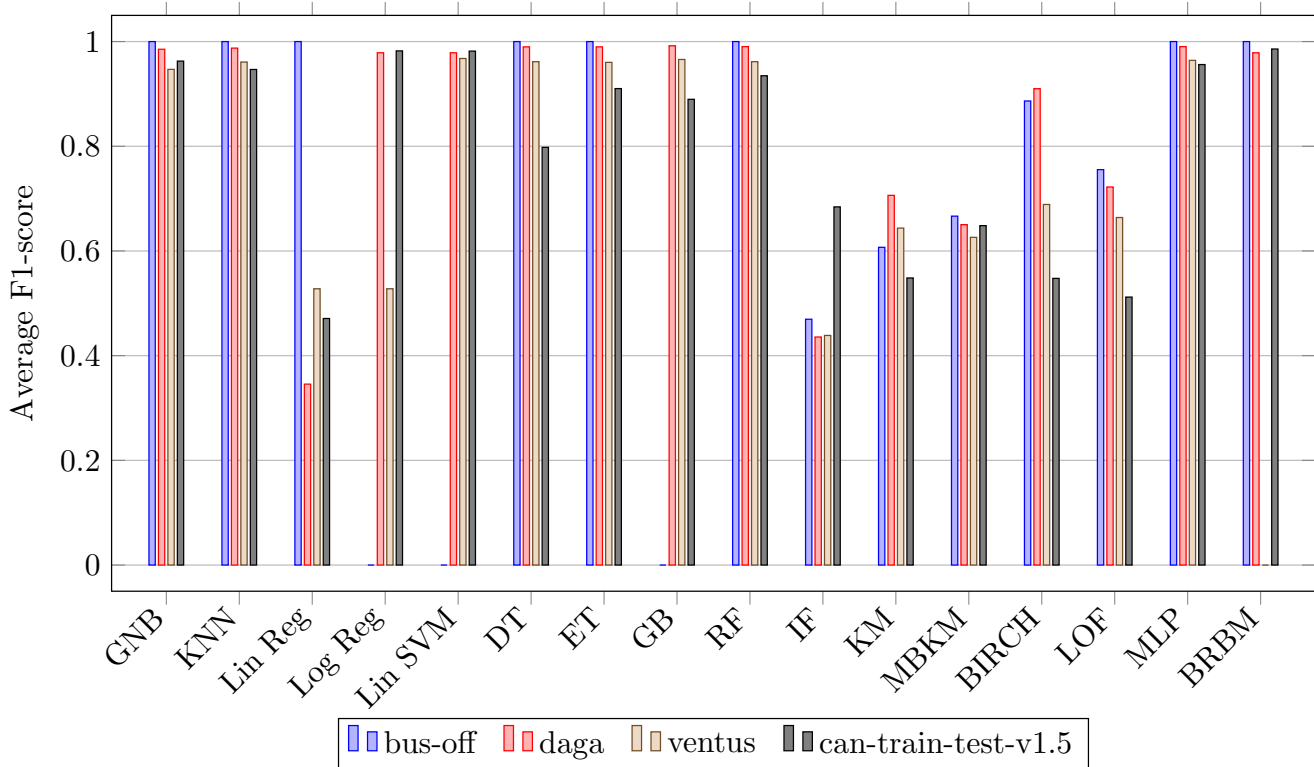


Fig. 3: Average F1-scores for bus-off, daga, ventus, and can-train-and-test-v1.5. Errors are set to zero. This graph corresponds to Tables 24, 25, 26, and 23, for the blue, red, tan, and black bars, respectively.

score, and G-mean, and perfect 0.0000s for false positive rate.

When we zoom out to look at all sixteen machine learning models (see Figure 3 and Table 24), we find that, when evaluated against the bus-off dataset, eight models achieved perfect results, five generated a lot of “false positives,” and three encountered an error. The Logistic Regression, Linear Support Vector Machine, and Gradient Boosting models all encountered an error; they all required *at least two classes but found only one (the attack-free class)*.

We expected these results; our evaluation depends on labeled data, and we cannot label suppress attacks—because messages are removed, not injected. To achieve perfect results, the models simply need to classify all the samples as “negative.” Therefore, our results are not indicative of the machine learning models’ ability to detect attacks. To properly assess an IDS’s ability to detect suppress attacks, we will need to significantly rethink and redesign our assessment strategies (see Section 8).

According to Table 24, when pitted against UNI-MORE Bus-Off, five models—Isolation Forest, K-Means, Mini-Batch K-Means, BIRCH, and Local Outlier Factor—generated a lot of false positives. While some might be genuine false positives, others might be the models’ reaction to the disappearance of the “suppressed” messages. These

five models are all unsupervised models, meaning they were never trained to classify all samples as “negative.” Instead, during training, these models developed an internal representation of attack-free or “normal” traffic. Then, during testing, they sought to identify deviations from “normal”—i.e., anomalies or attacks. These models might have recognized the suppress attacks as anomalies, but, since there were no actual attack messages, they would have been forced to label benign messages as attack messages. Interestingly, though we have been discussing *five* unsupervised machine learning models, there were actually *six* unsupervised models in our evaluation. The sixth model—Bernoulli Restricted Boltzmann Machine (BRBM)—achieved perfect results across all metrics.

The suppress attacks in can-train-and-test-v1.5 also yielded high F1-scores, though not as consistently (see “data-extended.xlsx,” an Excel spreadsheet, in the supplementary material). Each of can-train-and-test-v1.5’s four sub-datasets includes a suppress attack-specific testing subset, namely, test_05_suppress. We evaluated sixteen machine learning models against four suppress attack subsets for a total of 64 experiments. Of those 64 experiments, a mere seven resulted in perfect F1-scores. No machine learning model achieved perfect F1-scores for all four sub-datasets, though the BRBM model did achieve per-

fect F1-scores for sub-datasets #1, #2, and #3, running out of memory when pitted against sub-dataset #4.

In contrast, when evaluated against the bus-off dataset, eight machine learning models achieved perfect 1.0000 F1-scores for both `test_01_inhibition` and `test_02_shutdown` in `set_01`—and again in `set_02`. So, of bus-off’s 64 experiments, 32 resulted in perfect 1.0000 F1-scores. If we exclude the twelve experiments that failed due to the “at least two classes” error, then 32 of 52 experiments resulted in perfect F1-scores. The “at least two classes” error did not appear when evaluating the sixteen machine learning models against `can-train-and-test-v1.5` because the training data included both attack-free samples and attack samples (from non-suppress attacks). Although the training data contained non-suppress attacks, `test_05_suppress` did not.

The attack samples in `can-train-and-test-v1.5`’s training data, which range from DoS to fuzzing to spoofing and beyond, would have taught our machine learning models to look for a number of different attacks—not just suppress attacks. The difference in training data between `can-train-and-test-v1.5` and UNIMORE Bus-Off could explain the difference in overall performance between the two. In fact, the lower F1-scores associated with `can-train-and-test-v1.5` could indicate that our machine learning models successfully detected some of the suppress attacks. Since our label-based assessment strategies are incompatible with suppress attacks, even successful attack detection would be punished with lower F1-scores, not rewarded with higher F1-scores. An alternative—or perhaps complementary explanation—is that our machine learning models are trained to detect attacks, and, occasionally, they produce false positives. Some of the machine learning models that performed particularly poorly against suppress attacks have also performed poorly against attacks such as DoS, fuzzing, and spoofing. It is not surprising that these poor-performing models produce false positives during suppress attacks just as they do during DoS, fuzzing, and spoofing attacks.

7.3.2 `can-train-and-test-v1.5` vs. `daga`

The UNIMORE DAGA dataset does not contain suppress attacks; therefore, we can compare `daga` to `can-train-and-test-v1.5` by comparing Table 25 to Table 14 (the results for the suppress attack subsets were not averaged into Table 14). Figure 3 visually compares the two datasets in the form of a bar graph; the red bars represent the `daga` dataset, while the black bars represent the `can-train-and-test-v1.5` dataset.

As mentioned in Section 5.2, the UNIMORE DAGA dataset contains ample data. Even the highly trimmed and curated version of the `daga` dataset, found in `can-unimore-curated-v3`, contains about fifteen million training samples per sub-dataset—more samples than `set_01`, `set_03`,

and `set_04` of `can-train-and-test-v1.5`. As shown in Table 9, `set_01` of `daga` contains 14,947,920 training samples, whereas `set_01` of `can-train-and-test-v1.5` contains 11,460,705 training samples. Generally, a larger volume of training data leads to improved performance, as the machine learning model has more time to learn and more information to inform its decisions (i.e., classifications). All the experiments discussed in this section—and shown in Table 25, Table 27, Table 28, and Figure 3—were conducted against `can-unimore-curated-v3`’s `daga` dataset.

When we compare `daga` and `can-train-and-test-v1.5` model by model, we find that, for eleven machine learning models, `daga`’s F1-scores were better. For the remaining five models, `can-train-and-test-v1.5`’s F1-scores were better (see Figure 3).

During each experiment, one machine learning model is trained on one particular vehicle. Neither `daga` nor `can-train-and-test-v1.5` trains the model on more than one vehicle. However, `can-train-and-test-v1.5` tests the model on a second, *unknown* vehicle, whereas `daga` does not. `can-train-and-test-v1.5`’s “unknown vehicle” tests (`test_02_unknown_vehicle_known_attack` and `test_04_unknown_vehicle_unknown_attack`) demonstrate a model’s ability to generalize to unknown vehicles. If the model performs well, then we know that it can detect attacks against unknown vehicles without generating too many false positives. If the model performs poorly, then it has probably overfitted to the training vehicle.

Listing 3 Attack-free CAN traffic captured from a 2011 Chevrolet Traverse.

```
(1672163924.621479) 0C1#12E55F82124B979F
(1672163924.621724) 0C5#11D90EB31221AB2D
(1672163924.621942) 1E5#46FFD8C000002600
(1672163924.622330) 1F3#C0E0
(1672163924.624005) 0C7#02629A3D
(1672163924.624251) 0F9#07CD400000B828FF
(1672163924.624469) 189#0FFF0FFF300128FF
(1672163924.624722) 199#0FFF0E70F19000FF
(1672163924.628004) 1CB#100000
(1672163924.629932) 0C9#8409530D00010000
```

CAN traffic can vary widely between different vehicles, especially vehicles from different manufacturers—and vehicles with vastly different model years. For instance, Listing 3 presents attack-free CAN traffic from a 2011 Chevrolet Traverse, and Listing 4 provides analogous attack-free traffic from a 2017 Subaru Forester. In each case, the attack-free CAN traffic was collected via a CAN-to-USB device plugged into a laptop on one end and the vehicle’s On-Board Diagnostics port on the other end. The vehicle was driven normally, and all available CAN traffic was captured.

Listing 4 Attack-free CAN traffic captured from a 2017 Subaru Forester.

```
(1672865121.432848) 002#07317005AD000000
(1672865121.434040) 075#000000000009033B
(1672865121.434295) 0D0#E3F81700010400F5
(1672865121.434467) 0D1#5C000080
(1672865121.434720) 0D2#0000FFFF00010000
(1672865121.434935) 0D3#0005C00F602363
(1672865121.435172) 070#EF8400FFAB7F039A
(1672865121.435422) 080#1B800000427C033E
(1672865121.435668) 0D4#5700620054005F00
(1672865121.435878) 140#1401450500141AA0
```

It is evident, even in these short traces, that the Chevrolet Traverse’s arbitration IDs are entirely different from the Subaru Forester’s. In fact, no arbitration ID appears in both traces. For extended traffic captures, refer to Listings 5 and 6 in the appendix (Section A).

All of the UNIMORE datasets, including *daga*, were derived from a single vehicle: a 2016 Volvo V40 Kinetic. Consequently, when machine learning models are evaluated using *daga*, they are trained and tested on data from the same vehicle. In contrast, several testing scenarios in *can-train-and-test-v1.5* involve unknown vehicles that did not appear in the training data.

As Lampe and Meng explored in [53], overfitting can be a significant issue, especially when training on data from a single vehicle. While *daga* cannot expose such overfitting problems, *can-train-and-test-v1.5* is capable of revealing them by including data from multiple vehicles.

7.3.3 *can-train-and-test-v1.5* vs. *ventus*

The UNIMORE *Ventus* dataset contains two types of attacks—*injection* and *removal* (i.e., “*suppress*”). *Injection* attacks “*inject*” CAN messages, whereas *removal* attacks “*remove*” them (see Section 4.3). Consequently, *removal* attacks do not contain labeled attack samples—the attack is the removal itself. Because our assessment strategies depend on labeled attacks, we cannot accurately assess an IDS’s ability to detect *suppress* attacks. Therefore, we have excluded *ventus*’s *suppress* attacks from the results in Table 26 and the bar graph in Figure 3. Results with the *suppress* attacks included can be found in Table 32 in the appendix (see Section B). Similarly, for the *can-train-and-test-v1.5* dataset—Table 14 and Figure 3—we exclude the results of the *suppress* attacks (testing subset #5).

Looking at Figure 3 (and Tables 32 and 14), we can compare the average F1-scores between *ventus* and *can-train-and-test-v1.5*. We find that ten machine learning models performed better against *ventus*, whereas only five performed better against *can-train-and-test-v1.5*. We excluded the BRBM model from our analysis, since it per-

formed well against *can-train-and-test-v1.5* but ran out of memory when evaluated against *ventus*.

As discussed in Sections 6.5 and 7.4, sub-dataset #1 of the *ventus* dataset was specifically curated to exhibit train-test interdependence, which likely inflated the performance of some machine learning models. Referring back to the data (see “*data-extended.xlsx*,” an Excel spreadsheet, in the supplementary material), we compare *ventus*’s non-interdependent sub-dataset—sub-dataset #2—to *can-train-and-test-v1.5*. We find that train-test interdependence—or lack thereof—does not significantly impact our comparison between *ventus* and *can-train-and-test-v1.5*.

In fact, we find that eleven machine learning models performed better against *ventus*’s non-interdependent sub-dataset, whereas only four performed better against *can-train-and-test-v1.5*. *can-train-and-test-v1.5* compared more favorably against *ventus* before we excluded *ventus*’s interdependent sub-dataset. Removing this sub-dataset actually hurt *can-train-and-test-v1.5*’s relative performance. We conclude that while train-test interdependence had a significant impact on the machine learning models’ performance, it was not enough to overcome the significant performance gap between *ventus* and *can-train-and-test-v1.5*.

That said, sub-dataset #1 of *can-train-and-test-v1.5* contains 11,460,705 samples, encompassing a wide range of attacks, including DoS, fuzzing, systematic, and spoofing—to name a few (see Section 4.2). By contrast, sub-dataset #1 of *ventus* contains 27,270,266 samples, but only two attack types—*injection* and *removal*. As a consequence, our machine learning models have significantly more data to work with—per attack type—when trained on *ventus* than when trained on *can-train-and-test-v1.5*.

Furthermore, Table 26 excludes the *removal* attack, meaning that when our models are evaluated against *ventus*, they are only called upon to detect one type of attack. As discussed in Section 4.3, the *injection* attacks included in the *ventus* dataset are replay attacks; legitimate messages are captured and replayed at rates of 1, 10, 25, 50, or 100 messages per second. These replay attacks are consistent and extremely similar. It is relatively easy for an IDS to a single type of attack that comes in five variations. It is relatively difficult for an IDS to detect many *different* attacks with completely different patterns and signatures. These differences in training data, attack types, and attack consistency might help explain the stark performance differences between UNIMORE *Ventus* and *can-train-and-test-v1.5*.

Table 27: can-unimore-curated-v3, daga, set_01: Averages with default parameters.

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9947	0.9894	0.9947	0.9921	0.9973	0.0000
K -Nearest Neighbors	0.9895	0.9888	0.9895	0.9891	0.9921	0.0053
Linear Regression	0.2495	0.9814	0.2495	0.3821	0.2445	0.7602
Logistic Regression	0.9856	0.9718	0.9856	0.9786	0.9927	0.0000
Linear Support Vector Machine	0.9856	0.9718	0.9856	0.9786	0.9927	0.0000
Decision Tree	0.9908	0.9890	0.9908	0.9899	0.9934	0.0040
Extra Trees	0.9916	0.9891	0.9916	0.9903	0.9942	0.0032
Gradient Boosting	0.9943	0.9899	0.9943	0.9920	0.9968	0.0006
Random Forest	0.9917	0.9891	0.9917	0.9904	0.9943	0.0031
Isolation Forest	0.3088	0.9807	0.3088	0.4563	0.3045	0.6995
K-Means	0.5609	0.9663	0.5609	0.7078	0.5637	0.4335
Mini-Batch K-Means	0.4400	0.9808	0.4400	0.5975	0.4371	0.5656
BIRCH	0.9067	0.9711	0.9067	0.9376	0.9131	0.0804
Local Outlier Factor	0.5828	0.9823	0.5828	0.7065	0.5818	0.4191
Multi-Layer Perceptron	0.9947	0.9899	0.9947	0.9922	0.9974	0.0000
Bernoulli Restricted Boltzmann Machine	0.9856	0.9718	0.9856	0.9786	0.9927	0.0000

Table 28: can-unimore-curated-v3, daga, set_02: Averages with default parameters.

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9857	0.9718	0.9857	0.9786	0.9928	0.0000
K -Nearest Neighbors	0.9867	0.9856	0.9867	0.9860	0.9908	0.0051
Linear Regression	0.1962	0.9822	0.1962	0.3091	0.1905	0.8148
Logistic Regression	0.9857	0.9718	0.9857	0.9786	0.9928	0.0000
Linear Support Vector Machine	0.9857	0.9718	0.9857	0.9786	0.9928	0.0000
Decision Tree	0.9908	0.9892	0.9908	0.9899	0.9933	0.0042
Extra Trees	0.9900	0.9889	0.9900	0.9894	0.9925	0.0049
Gradient Boosting	0.9943	0.9898	0.9943	0.9920	0.9969	0.0006
Random Forest	0.9919	0.9892	0.9919	0.9905	0.9944	0.0030
Isolation Forest	0.2752	0.9823	0.2752	0.4149	0.2704	0.7342
K-Means	0.5570	0.9652	0.5570	0.7047	0.5600	0.4369
Mini-Batch K-Means	0.5550	0.9652	0.5550	0.7031	0.5580	0.4389
BIRCH	0.8101	0.9698	0.8101	0.8824	0.8156	0.1787
Local Outlier Factor	0.6003	0.9822	0.6003	0.7380	0.5988	0.4027
Multi-Layer Perceptron	0.9912	0.9867	0.9912	0.9887	0.9952	0.0008
Bernoulli Restricted Boltzmann Machine	0.9857	0.9718	0.9857	0.9786	0.9928	0.0000

7.4 Train-Test Interdependence

Train-test interdependence becomes a problem when the training data is *not* completely independent of the testing data. Train-test interdependence ties into the problem of data leakage (also known as “pattern leakage”). Information from testing data “leaks” into training data, which, in turn, leads to overfitting and inflated performance metrics.

Bouke and Abdullah [5] pitted six machine learning models— k -Nearest Neighbors, Logistic Regression, support vector machine, Decision Tree, Gradient Boosting, and Random Forest—against three intrusion detection datasets—NSL-KDD [137, 138], UNSW-NB15 [88, 89], and KDD Cup 1999 [38]. To investigate the impact of pattern leakage (i.e., train-test interdependence) on performance, they constructed two versions of each dataset—one with pattern leakage and one without. Unsurprisingly, they found that pattern leakage inflated accuracy, and that the Decision Tree and Gradient Boosting models were significantly more

susceptible to pattern leakage-inflated performance than the others. The Random Forest model was particularly robust in the face of pattern leakage.

Bouke and Abdullah used the Python programming language to build the driver program, leveraging `pandas` [94] to process and manipulate the data and `scikit-learn` [96] to implement the machine learning models. We also wrote our driver program in Python, used `pandas` to load and pre-process our data, and used `scikit-learn` to instantiate our sixteen machine learning models. Our approach to train-test interdependence is remarkably similar to Bouke and Abdullah’s—especially with regard to experimental setup and software choices. Additionally, the UNIMORE DAGA and UNIMORE Ventus datasets are both intrusion detection datasets—albeit automotive-specific intrusion detection datasets.

Figure 4 visualizes the results of experiments run against `set_01` and `set_02` of the UNIMORE DAGA dataset in the form of a bar graph. The blue and red bars corre-

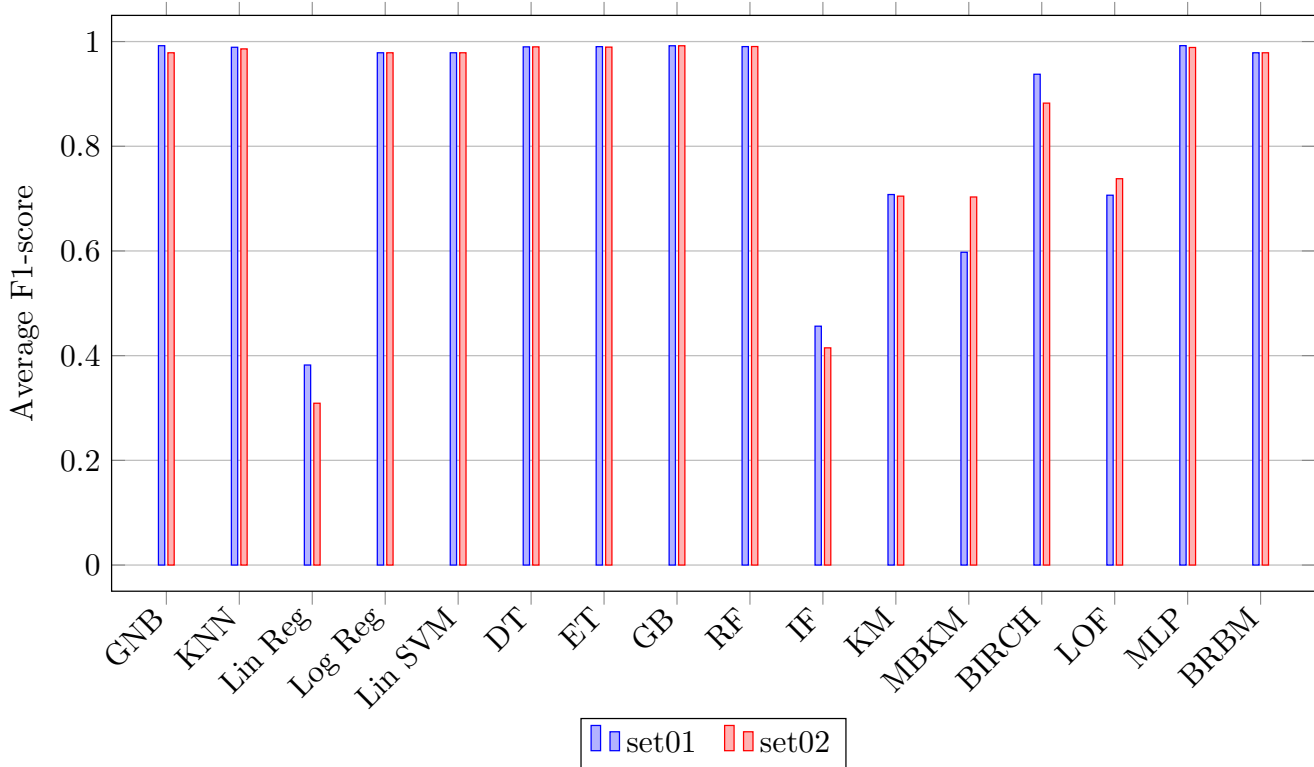


Fig. 4: Average F1-scores for set_01 (interdependent) and set_02 (non-interdependent) of the daga dataset. This graph corresponds to Tables 27 and 28, for the blue and red bars, respectively.

spond to the average F1-scores achieved by the models when pitted against set_01 (interdependent) and set_02 (non-interdependent), respectively.

Interestingly, our results differ quite significantly from [5]. When pitted against the daga dataset, many of our machine learning models perform so similarly—in terms of F1-score—on the interdependent sub-dataset (set_01) and the non-interdependent sub-dataset (set_02) that we cannot visually discern a difference between the blue and red bars. The Log Reg, Lin SVM, DT, ET, GB, RF, and BRBM models are especially indiscernible on the bar graph. Some models, such as Extra Trees and Random Forest, performed slightly differently between the two sub-datasets. Others, such as Decision Tree and Gradient Boosting, achieved equal F1-scores on the interdependent and non-interdependent sub-datasets.

Table 27 outlines the performance of our sixteen machine learning models against set_01 of the daga dataset. For each model, the results are averaged across all six testing subsets, that is,

1. test_01_DoS
2. test_02_arbitration_id_fuzzing
3. test_03_data_field_fuzzing
4. test_04_single_arbitration_id_replay
5. test_05_arbitrary_sequence_replay

6. test_06_ordered_sequence_replay

Similarly, Table 28 summarizes the performance of each of our machine learning models against set_02 of the daga dataset. Comparing F1-scores between Table 27 (interdependent) and Table 28 (non-interdependent), we find that eight of our sixteen machine learning models performed better against the interdependent sub-dataset, five performed equally well on both sub-datasets, and three performed worse—Random Forest, Mini-Batch K-Means, and Local Outlier Factor. The performance difference for the Random Forest model was minute: 0.0001. The performance differences were much more pronounced for Mini-Batch K-Means and Local Outlier Factor: the former jumped from 0.5975 on the interdependent sub-dataset to 0.7031 on the non-interdependent sub-dataset, the latter climbed from 0.7065 to 0.7380. Both Mini-Batch K-Means and Local Outlier Factor are *unsupervised* models.

Next, we turn to the ventus dataset’s “injection” attack (we exclude the “removal” attack because it is uninformative). Since the Bernoulli Restricted Boltzmann Machine ran out of memory during both experiments, we have fifteen machine learning models to look at, not sixteen. Of those fifteen machine learning models, twelve performed better against set_01 (interdependent) than set_02 (non-interdependent). No model performed equally against both

sub-datasets, but three performed worse against `set_01` than `set_02`. All results for the `ventus` dataset are available in “`data-extended.xlsx`,” an Excel spreadsheet, in the supplementary material.

Interestingly, the three models that deviated from the trend were all *unsupervised* models, and, more specifically, they were all *clustering* models—K-Means, Mini-Batch K-Means, and BIRCH. Our evaluation included only three clustering models, and all three both (1) performed poorly on `set_01` (interdependent) and (2) performed significantly better on `set_02` (non-interdependent). When pitted against the interdependent sub-dataset, K-Means achieved an F1-score of 0.5864, Mini-Batch K-Means achieved 0.5510, and BIRCH achieved 0.4551. Against the non-interdependent sub-dataset, the F1-scores were 0.7010, 0.7009, and 0.9224—significantly higher than before.

These results also suggest that train-test interdependence particularly inflates the performance of high-performing models—models that achieve F1-scores of 0.9 or higher for both sub-datasets. For the `ventus` dataset, eight machine learning models were “high-performing models” by this definition. All eight of these models achieved even higher performance with the advantage of train-test performance; in other words, all eight of these models performed better against `set_01` than `set_02`. We theorize that, in general, the low-performing models struggled to capture essential data patterns during training; therefore, when pitted against `set_01`, they failed to capitalize on the “leaked” patterns available in the training data.

7.5 Standard Deviation

In our evaluation, we focused on evaluating the six datasets, not the sixteen machine learning models—because this work constitutes a dataset evaluation, not an evaluation of machine learning models or IDSs. However, CAN intrusion detection datasets are designed to train and *test* IDSs. They are designed to assess the performance of IDS. Researchers use datasets to quantify the performance of an IDS, and, importantly, they use datasets to *compare* different IDSs. CAN intrusion detection datasets must distinguish between “good” IDSs and “bad” IDSs. More than that, however, an effective dataset should be able to differentiate between “good” models, “great” models, and “outstanding” models. One of the problems Rajapaksha et al. [103] found with the HCLR Car Hacking dataset was its lack of differentiation power; a number of works used `hcr1-ch` to evaluate a proposed IDS, and in most cases, the proposed IDS achieved a 0.99+ F1-score for all attacks (see Section 4.1).

To measure each dataset’s differentiation power, we leveraged standard deviation (STD), which captures the variation of a value relative to the mean. Low standard deviation indicates that values cluster near the mean, whereas

high standard deviation indicates a wider range. In the context of CAN intrusion detection datasets, wider ranges are associated with greater differentiation power. A dataset with a high standard deviation should magnify slight differences in capability between two IDSs, so that researchers can see which one is superior.

We calculated the standard deviation of the machine learning models’ F1-scores to assess how much the F1-score varied from model to model. For `can-train-and-test-v1.5`, the STD was 0.2392 (excluding suppress attacks). For `hcr1-ch`, the STD was 0.2254; for `hcr1-sa`, the STD was 0.2333. Looking at `can-unimore-curated-v3`, we find that the standard deviations were 0.1824 for UNIMORE Bus-Off, 0.2121 for UNIMORE DAGA, and 0.2100 (excluding suppress attacks) for UNIMORE Ventus. We can see that `can-train-and-test-v1.5` has the highest standard deviation, followed by the HCRL datasets and then the UNIMORE datasets.

As mentioned earlier, our experimental setup is ill-suited to suppress (or “removal”) attacks. Many of our machine learning models are supervised and depend on labeled data; additionally, we leverage labeled data to assess our models’ performance. Therefore, we exclude the results of the suppress attacks—which are uninformative—from the standard deviations for the `can-train-and-test-v1.5` dataset and the `ventus` dataset. We cannot exclude suppress attacks from the `bus-off` dataset because it contains only suppress attacks. That said, the `bus-off` dataset’s STD is uninformative; our machine learning models can obtain perfect F1-scores on suppress attack traces by deciding “negative” for all samples. Indeed, as discussed in Section 7.3, eight of our sixteen machine learning models averaged perfect metrics—accuracy, precision, recall, F1-score, G-mean, and FPR—when pitted against the `bus-off` dataset (see Table 24). The eight 1.0000 F1-scores drastically reduce the standard deviation of the `bus-off` dataset.

As discussed in Section 4.1, the HCRL datasets have a number of weaknesses, and the HCRL Survival Analysis dataset, in particular, lacks adequate training data—especially attack-free data. We were surprised to find that `hcr1-sa` had a higher standard deviation than the UNIMORE DAGA and UNIMORE Ventus datasets. However, upon reviewing the data, we suspect that `hcr1-sa` high STD is more the result of poor performance than differentiation power. If our machine learning models are underfitting to `hcr1-sa`, then they are not training to the point of optimization. Instead, our machine learning models might be guided by the randomness with which they were initialized [116]. Such subpar models would be less consistent than adequately trained models, leading to `hcr1-sa`’s high STD. While standard deviation is a useful metric, it only tells part of the story.

8 Limitations & Future Work

8.1 Limitations

We have identified two main limitations of our work—both of which correspond to opportunities for future work:

1. *Our feature investigation is not comprehensive.* In our future work, we will select additional features for investigation—e.g., data length code (DLC), time delta (time difference between two successive messages, also known as the “inter-message interval”). Moreover, for each machine learning model, we will determine which configuration of features is optimal.
2. *Our dataset comparison is not comprehensive.* In our future work, we will target additional CAN intrusion detection datasets for comparison (e.g., signal-based datasets). In addition, we will update our experimental setup, machine learning models, and evaluation strategies so that we can accurately evaluate a machine learning model’s ability to detect suppress attacks. Furthermore, we will analyze the use cases of the existing open-access CAN intrusion detection datasets, and we will determine, for each use case, which dataset would be best. For example, for signal-based IDSs, a signal-based dataset would be better than `can-train-and-test-v1.5`. For more information about existing open-access datasets, including features, advantages & disadvantages, use cases, and guidance, we refer the reader to [67] (Sections 3.1, 6, and 8), [146] (Sections 3 and 4), and [103] (Section 5.5).

8.2 Future Work

Feature investigation. We have identified three features for further investigation: (1) the data length code (DLC) feature, (2) the data field feature, and (3) the time delta feature.

As mentioned in Section 6.3, the DLC is included as a feature in both the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset. As such, we have developed the following research question: Would the DLC feature enhance the performance of one or more of our machine learning models?

Typically, the DLC feature should correspond to the actual length of the data field feature. While attack attempts using an invalid DLC would be immediately detected by CAN controllers, such discrepancies may indicate ongoing reconnaissance, fuzzing, or system probing. An attacker could send a frame with a DLC indicating eight bytes but transmit fewer bytes, exploring communication protocol vulnerabilities or system responses. Moreover, DLC mismatches could signal system malfunctions or incomplete

data transmissions. However, even a moderately-capable attacker should be able to generate attack CAN frames in which the DLC is correct for the attack CAN frame’s data field. `cansend`, one of Linux’s `can-utils`, automatically computes and sends the correct value for the DLC based on the data field. In fact, depending on the tools and technologies used, it might be more difficult to send a CAN frame with an invalid DLC. Consequently, the predictive value of the DLC feature for machine learning models remains uncertain and warrants further investigation.

In this work, we investigated the `data field` feature—specifically, we investigated the impact of subdividing (or not subdividing) the data field into eight discrete features (one feature per byte). However, we did not explore the impact of removing the `data field` feature in its entirety. Some IDSs (e.g., ID sequence-based IDSs [79, 62, 65]) do not look at the `data field` feature. Instead, they analyze patterns in the `arbitration ID` feature. Therefore, we have composed the following research question: If we remove the `data field` feature, will some machine learning models perform better—perhaps because noise in the data has been reduced?

The `time delta` feature is the time difference between two consecutive messages—also known as the “inter-message interval” or “elapsed time.” This feature might prove more informative than the `timestamp` feature. As mentioned in Section 6.1, `can-train-and-test-v1.5`’s `timestamp` has been standardized such that all traces begin at 2023-01-01 00:00:00.000000000 (January 1, 2023, at midnight). This standardization mitigates noise arising from the dates and times that traces happened to be collected. For example, if many attack-free traces were collected on January 12th, and many attack traces were collected on January 13th, then the IDS might wrongly associate attacks with January 13th, generating false positives for attack-free traffic on January 13th and missing attacks that occur on other days. However, standardization cannot entirely eliminate timestamp-related noise. If, in a timestamp-standardized dataset, many attacks happen to occur later in the trace, then the IDS might wrongly associate later timestamps with attacks, missing earlier attacks and generating false positives for later attack-free traffic. We plan to build upon earlier work with the `time delta` feature by Kidmose and Meng [53], who experimented with the original `can-train-and-test` dataset and a limited number of machine learning models.

Dataset comparison. Though `hcr1-ch` is believed to be the most popular CAN intrusion detection dataset in the literature [103, 67], many others are also wildly popular or gaining traction. For example, the Synthetic CAN Bus dataset (SynCAN) [36] is the most popular CAN intrusion detection dataset when it comes to evaluating unsupervised payload-based IDSs [103]. Moreover, the Real

ORNL Automotive Dynamometer CAN intrusion (ROAD) dataset [146, 145] contains 33 attack traffic captures—and ten unique attacks, depending on how the unique attacks are counted [67]—compared to the ≤ 5 attacks available in pre-existing open-access CAN intrusion detection datasets. The `can-train-and-test` dataset contains only nine unique attacks; the extended dataset—`can-train-and-test-v1.5`—contains eleven. Unfortunately, while the ROAD dataset contains descriptions that can help researchers label the CAN traffic captures, the CAN traffic captures themselves are not labeled. In order to compare the ROAD dataset to the `can-train-and-test-v1.5` dataset, we would need to label the ROAD dataset. In our future work, we plan to label the ROAD dataset and compare it to `can-train-and-test-v1.5`. In addition, we plan to develop “practitioner guidance” indicating which dataset—`can-train-and-test-v1.5` or ROAD—would be best suited to a given research problem.

At present, we rely on labeled attack-free and attack data to quantify the performance of a machine learning model. Unfortunately, suppress attacks, by nature, contain attack-free samples but no attack samples—the attack is the suppression (i.e., removal) of legitimate messages. As such, when we evaluate an IDS’s performance during a suppress attack, the metrics we obtain are uninformative; an IDS can obtain perfect metrics simply by deciding “negative” for all samples. To accurately evaluate suppress attacks, we need to redesign our experiments and rethink our evaluation strategies. Previous works, e.g., [131, 98, 36], have leveraged predefined algorithms or unsupervised machine learning to detect suppress attacks. Six of our machine learning models were unsupervised models; we could still use those models if we developed different evaluation strategies. Alternatively, we could explore strategies that would allow us to “label” suppress attacks. For example, our assessment data (i.e., the “correct answers” that allow us to check an IDS’s decisions for accuracy) could contain null samples with “-1” as the label. Our IDSs could be configured to add “-1” samples wherever they detect a suppressed message.

As mentioned earlier, we did not compare `can-train-and-test-v1.5` to a signal-based dataset. In our future work, we plan to conduct a *qualitative* assessment of the benefits of signal-based datasets (e.g., SynCAN, ROAD) compared to raw datasets (e.g., `hcr1-sa`, `can-train-and-test-v1.5`). In a later step, we would like to conduct a *quantitative* assessment of signal-based datasets vs. raw datasets. We will investigate the possibility of adapting our machine learning models to receive either raw data or signal values as input. Such an adaptation would allow us to quantitatively compare `can-train-and-test-v1.5` to a signal-based dataset such as SynCAN.

During our paper’s revision process, we identified several significant automotive intrusion detection datasets that

were previously unavailable or unknown during our initial experiments. These datasets are now documented in Section 2.2, specifically in Tables 1, 2, and 3. Two datasets warrant particular attention: the Transmission-Resuming Time-Based Intrusion Detection System (TTIDS) dataset [72, 74] and the CAN-MIRGU dataset [106, 104, 105]. TTIDS represents a milestone as the first dataset to capture masquerade attacks using diagnostic services and bus-off attacks on actual vehicles, although its narrow focus on masquerade attacks somewhat constrains its applicability. The CAN-MIRGU dataset, generated from a 2016 electric vehicle, presents a broader spectrum of attacks including denial of service, fuzzing, replay, spoofing, suspension, masquerade, and concurrent attack scenarios. Several CAN-MIRGU attacks demonstrated direct impact on safety-critical vehicle control systems, e.g., stiffening or loosening the steering wheel. Both datasets feature comprehensive labeling, enhancing their value for research. While detailed comparative analysis of these datasets against older, more popular datasets remains as future work, we include their documentation here and in Section 2.2 to facilitate their discovery and utilization by the research community.

9 Conclusion

In this paper, we evaluated six different automotive intrusion detection datasets:

1. HCRL Car Hacking (`hcr1-ch`)
2. HCRL Survival Analysis (`hcr1-sa`)
3. `can-train-and-test-v1.5`
4. UNIMORE Bus-Off (`bus-off`)
5. UNIMORE DAGA (`daga`)
6. UNIMORE Ventus (`ventus`)

Specifically, we conducted a feature evaluation of the newly-published `can-train-and-test-v1.5` dataset, and we analyzed train-test interdependence in the context of the three UNIMORE datasets—with an emphasis on the UNIMORE DAGA dataset. In addition, we pitted `can-train-and-test-v1.5` against two well-established automotive intrusion detection datasets, namely, HCRL Car Hacking and HCRL Survival Analysis. Ultimately, we compared the newly-published, promising `can-train-and-test-v1.5` dataset with some of its fellow up-and-coming datasets, in particular, the three UNIMORE datasets—Bus-Off, DAGA, and Ventus.

In our feature sleuthing, we determined that the presence of the `timestamp` feature enhanced the performance of many of our machine learning models. In particular, if a given model was even moderately successful (i.e., an average F1-score greater than 0.75), then excluding the `timestamp` either had no impact or had a negative impact. As

such, we determined that there is no benefit in removing the `timestamp` feature. When it came to the `data field` feature, we found that some machine learning models benefited from a subdivided data field (eight bytes as distinct features), while others benefited from an unpartitioned data field. Therefore, we can choose to subdivide—or not subdivide—the `data field` feature depending on the specific model selected.

An analysis of train-test interdependence in the UNIMORE datasets confirmed that, as expected, train-test interdependence significantly inflated performance—especially for high-performing machine learning models. For the UNIMORE DAGA dataset, out of our sixteen machine learning models, thirteen performed equally well or better on the train-test interdependent (i.e., advantaged) sub-dataset than on the non-interdependent sub-dataset. Only three performed worse on the interdependent sub-dataset. When we focused on high-performing machine learning models (which earned F1-scores of 9.0 or higher on both sub-datasets), we found that nine out of ten machine learning models performed equally well or better on the interdependent sub-dataset. For the UNIMORE Ventus dataset, the numbers were even more compelling. When pitted against the “injection” attack, twelve out of fifteen machine learning models performed *better*—not equally well or better—against the train-test interdependent sub-dataset than the non-interdependent sub-dataset. Again, only three performed worse on the interdependent sub-dataset. One machine learning model ran out of memory and was excluded from our analysis. There were eight high-performing machine learning models for the UNIMORE Ventus dataset—all of them performed better on the interdependent sub-dataset than the non-interdependent one.

In our comparison, we concluded that `can-train-and-test-v1.5` provides novel features and insights when juxtaposed against existing open-access CAN intrusion detection datasets—such as the HCRL datasets. Specifically, we observed that `can-train-and-test-v1.5` contains much more training and testing data than either of the HCRL datasets (an order of magnitude more), and `can-train-and-test-v1.5`'s attack traffic captures are more realistic in terms of class imbalance. When pitted against the `can-train-and-test-v1.5` dataset, half of our machine learning models—eight out of sixteen—achieved an average F1-score above 0.9. When pitted against the HCRL datasets, which contain much less training data, the average F1-scores were noticeably lower. Though average F1-scores were high for the `can-train-and-test-v1.5` dataset, there was no loss of differentiation power; the standard deviation of the models' F1-scores was 0.2392 (excluding suppress attacks), exceeding the standard deviations of the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset—0.2254 and 0.2333, respectively.

When we pitted `can-train-and-test-v1.5` against the three UNIMORE datasets, in terms of both dataset quality and machine learning model performance, the race was much tighter. `can-train-and-test-v1.5` has the advantage of authentic attacks conducted in a real vehicle under driving conditions. However, UNIMORE DAGA's attacks (as well as UNIMORE Ventus's injection attacks) were constructed using high-fidelity simulations (complete with CAN transceivers and resistors). In addition, UNIMORE DAGA demonstrates extreme class imbalance—even exceeding `can-train-and-test-v1.5`—which is much more realistic to automotive intrusion detection than balanced classes. The UNIMORE Bus-Off dataset's standard deviation was 0.1824, which is low, but is also uninformative because we could not accurately measure the performance of our machine learning models against suppress attacks. The standard deviations of the DAGA and Ventus datasets were 0.2121, and 0.2100 (excluding suppress attacks), respectively. We found that `can-train-test-v1.5`, UNIMORE DAGA, and UNIMORE Ventus—and, in some cases, even UNIMORE Bus-Off—are well suited to automotive intrusion detection applications. These datasets provide ample quantities of high-fidelity data, a variety of attacks, and requisite differentiation power. We recommend them to practitioners in automotive intrusion detection and related domains.

10 Author Information

10.1 Contributions

B.K.: Conceptualization, Data curation, Investigation, Methodology, Writing – original draft, Writing – review & editing. **A.K.:** Formal analysis, Visualization, Writing – review & editing. **W.M.:** Methodology, Supervision, Writing – review & editing.

11 Declarations

11.1 Competing Interests

The authors have no relevant financial or non-financial interests to disclose.

Acknowledgements We thank the reviewers for their detailed comments, which greatly improved the quality of this manuscript.

We thank David and Kawa of 24 Hour Auto Repair of Lincoln, NE, for their help in sourcing modern test vehicles and facilitating industry contacts. We thank Ster Auto of Lincoln, NE, for providing access to vehicles for our experiments.

We also thank our test drivers, especially Lars Brasen, Marius Brasen, Christian Lampe, Julie Lampe, Ryan Lampe, and Wayne & Vickie Olsen, who dedicated their time to carefully execute specific

driving scenarios and test conditions, enabling us to collect comprehensive driving data.

References

1. Alfaridus A, Rawat DB (2021) Intrusion detection system for can bus in-vehicle network based on machine learning algorithms. 2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON) URL <https://ieeexplore.ieee.org/document/9666745>
2. Alkhatib N, Mushtaq M, Ghauch H, Danger JL (2022) Unsupervised network intrusion detection system for avtp in automotive ethernet networks. 2022 IEEE Intelligent Vehicles Symposium (IV) DOI 10.1109/IV51971.2022.9827285, URL <https://ieeexplore.ieee.org/document/9827285>
3. Bi Z, Xu G, Xu G, Wang C, Zhang S (2022) Bit-level automotive controller area network message reverse framework based on linear regression. *Sensors* 22:981, DOI 10.3390/s22030981, URL <https://www.mdpi.com/1424-8220/22/3/981>
4. Bilge L, Dumitraş T (2012) Before we knew it: an empirical study of zero-day attacks in the real world. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Association for Computing Machinery, p 833 – 844, DOI 10.1145/2382196.2382284, URL <https://dl.acm.org/doi/10.1145/2382196.2382284>
5. Bouke MA, Abdullah A (2023) An empirical study of pattern leakage impact during data preprocessing on machine learning-based intrusion detection models reliability. *Expert Systems with Applications* 230:120,715, DOI 10.1016/j.eswa.2023.120715, URL <https://www.sciencedirect.com/science/article/pii/S0957417423012174>
6. Boumiza S, Braham R (2017) Intrusion threats and security solutions for autonomous vehicle networks. 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA) URL <https://ieeexplore.ieee.org/document/8308273>
7. Bozdal M, Samie M, Aslam S, Jennions I (2020) Evaluation of can bus security challenges. *Sensors* 20, URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7219335/>
8. Burakova Y, Hass B, Millar L, Weimerskirch A (2016) Truck hacking: An experimental analysis of the sae j1939 standard. WOOT'16: Proceedings of the 10th USENIX Conference on Offensive Technologies URL <https://www.usenix.org/system/files/conference/woot16/woot16-paper-burakova.pdf>
9. CAN in Automation (CiA) (2023) Can fd - the basic idea. CAN in Automation (CiA) URL <https://www.can-cia.org/can-knowledge/can/can-fd/>
10. Checkoway S, McCoy D, Kantor B, Anderson D, Shacham H, Savage S, Koscher K, Czeskis A, Roesner F, Kohno T (2011) Comprehensive experimental analyses of automotive attack surfaces. In: *20th USENIX Security Symposium (USENIX Security 11)*, USENIX Association, URL <https://www.usenix.org/conference/usenix-security-11/comprehensive-experimental-analyses-automotive-attack-surfaces>
11. Control TB, Committee CN (2022) Can fd network security: J1939-91c. SAE Standards URL <https://www.sae.org/standards/content/j1939-91c/>
12. Doan TP, Ganesan S (2017) Can crypto fpga chip to secure data transmitted through can fd bus using aes-128 and sha-1 algorithms with a symmetric key. SAE International URL <https://saemobilus.sae.org/content/2017-01-1612/>
13. Du J, Tang R, Feng T (2022) Security analysis and improvement of vehicle ethernet some/ip protocol. *Sensors* 22:6792 – 6810, URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9503536/>
14. Dupont G, Lekidis A, den Hartog J, Etalle S (2019) Automotive controller area network (CAN) bus intrusion dataset v2. 4TU-Centre for Research Data DOI 10.4121/uuid:b74b4928-c377-4585-9432-2004dfa20a5d, URL https://data.4tu.nl/articles/_/12696950/2
15. Eichensehr KE (2022) Ukraine, cyberattacks, and the lessons for international law. *AJIL Unbound* 116:145 – 149, DOI 10.1017/aju.2022.20, URL <https://www.cambridge.org/core/journals/american-journal-of-international-law/article/ukraine-cyberattacks-and-the-lessons-for-international-law/69B36016B06998BCE1EC67C757CDF34D>
16. Foruhandeh M, Man Y, Gerdes R, Li M, Chantem T (2019) Simple can bus voltage dataset. GitHub URL <https://github.com/harry1993/simple-dataset>
17. Foruhandeh M, Man Y, Gerdes R, Li M, Chantem T (2019) SIMPLE: single-frame based physical layer identification for intrusion detection and prevention on in-vehicle networks. *ACSAC '19: Proceedings of the 35th Annual Computer Security Applications Conference* pp 229 – 244, DOI 10.1145/3359789.3359834, URL <https://dl.acm.org/doi/10.1145/3359789.3359834>
18. Foster I, Koscher K (2015) Exploring controller area networks. ;login: 40:6 – 10, URL <https://www.usenix.org/system/files/>

- login/articles/login_dec15_02_foster.pdf
19. Gazdag A (2024) CrySyS dataset of CAN traffic logs containing fabrication and masquerade attacks. CrySyS Blog URL <https://blog.crysys.hu/2024/04/crysys-dataset-of-can-traffic-logs-containing-fabrication-and-masquerade-attacks/>
 20. Gazdag A, Ferenc R, Buttyán L (2023) CrySyS dataset of CAN traffic logs containing fabrication and masquerade attacks. *Scientific Data* 10:903, DOI 10.1038/s41597-023-02716-9, URL <https://www.nature.com/articles/s41597-023-02716-9>
 21. Gazdag A, Ferenc R, Buttyán L (2023) CrySyS dataset of CAN traffic logs containing fabrication and masquerade attacks. figshare DOI 10.6084/m9.figshare.c.6726165.v1, URL https://springernature.figshare.com/collections/CrySyS_dataset_of_CAN_traffic_logs_containing_fabrication_and_masquerade_attacks/6726165/1
 22. Gholamy A, Kreinovich V, Kosheleva O (2018) Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation. University of Texas at El Paso URL https://scholarworks.utep.edu/cs_techrep/1209/
 23. Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press, <http://www.deeplearningbook.org>
 24. Greenberg A (2015) Gm took 5 years to fix a full-takeover hack in millions of onstar cars. *Wired* URL <https://www.wired.com/2015/09/gm-took-5-years-fix-full-takeover-hack-millions-onstar-cars/>
 25. Greenberg A (2015) Hackers remotely kill a jeep on the highway—with me in it. *Wired* URL <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>
 26. Greenberg A (2017) How an entire nation became russia's test lab for cyberwar. *Wired* URL <https://www.wired.com/story/russian-hackers-attack-ukraine/>
 27. Greenberg A (2018) The untold story of notpetya, the most devastating cyberattack in history. *Wired* URL <https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/>
 28. Grimm D, Stang M, Sax E (2021) Context-aware security for vehicles and fleets: A survey. *IEEE Access* 9:101,809 – 101,846, URL <https://ieeexplore.ieee.org/document/9483907>
 29. Groza B (2024) ECUPrint (datasets). Universitatea Politehnica Timisoara: Departamentul de Automatică și Informatică Aplicată URL <https://www.aut.upt.ro/~bgroza/projects/ecuprint/>
 30. Groza B, Murvay S, van Herrewege A, Verbauwheide I (2012) Libra-can: A lightweight broadcast authentication protocol for controller area networks. *International Conference on Cryptology and Network Security* URL https://link.springer.com/chapter/10.1007/978-3-642-35404-5_15
 31. Guerra L, Xu L, Bellavista P, Chapuis T, Duc G, Mozharovskiy P, Nguyen VT (2024) AI-driven intrusion detection systems (IDS) on the ROAD dataset: A comparative analysis for automotive controller area network (CAN). *CSCS '24: Proceedings of the 2024 Cyber Security in CarS Workshop* pp 39 – 49, DOI 10.1145/3689936.3694696, URL <https://dl.acm.org/doi/10.1145/3689936.3694696>
 32. Hacking and Countermeasure Research Lab (2019) Intrusion detection for automotive [AUTO-TRANSLATED FROM KOREAN]. *K-Cyber Security Challenge* URL <http://datachallenge.kr/challenge19/convergence-security/vehicle/introduction/>
 33. Han ML, Kwak BI, Kim HK (2018) Anomaly intrusion detection method for vehicular networks based on survival analysis. *Vehicular Communications* 14:52 – 63, DOI 10.1016/j.vehcom.2018.09.004, URL <https://www.sciencedirect.com/science/article/pii/S2214209618301189>
 34. Han ML, Kwak BI, Kim HK (2018) Survival analysis dataset for automobile IDS. *Hacking and Countermeasure Research Lab* URL <https://ocslab.hksecurity.net/Datasets/survival-ids>
 35. Han ML, Kwak BI, Kim HK (2019) In-vehicle network intrusion detection challenge. *Hacking and Countermeasure Research Lab* URL <https://ocslab.hksecurity.net/Datasets/datachallenge2019/car>
 36. Hanselmann M, Strauss T, Dormann K, Ulmer H (2020) CANet: An unsupervised intrusion detection system for high dimensional CAN bus data. *IEEE Access* 8:58,194 – 58,205, DOI 10.1109/ACCESS.2020.2982544, URL <https://ieeexplore.ieee.org/document/9044377>
 37. Harvey J, Kumar S (2020) A survey of intelligent transportation systems security: Challenges and solutions. *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)* URL <https://ieeexplore.ieee.org/document/9123012>
 38. Hettich S, Bay SD (1999) The uci kdd archive. University of California, Irvine URL <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

39. Hossain MA, Islam MS (2023) Ensuring network security with a robust intrusion detection system using ensemble-based machine learning. *Array* 19:100,306, DOI 10.1016/j.array.2023.100306, URL <https://www.sciencedirect.com/science/article/pii/S2590005623000310>
40. Influx Technology (2021) CAN bus: Baud rate and its calculation. Influx Technology URL <https://www.influxtechnology.com/post/ baud-rate>
41. Islam R, Refat RUD (2020) Improving can bus security by assigning dynamic arbitration ids. *Journal of Transportation Security* 13:19 – 31, URL <https://link.springer.com/article/10.1007/s12198-020-00208-0>
42. Jeong S, Lee S, Lee H, Kim HK (2023) X-CANIDS dataset (in-vehicle signal dataset). Hacking and Countermeasure Research Lab URL <https://ocslab.hksecurity.net/Datasets/x-canids-dataset-in-vehicle-signal-dataset>
43. Jeong S, Lee S, Lee H, Kim HK (2023) X-CANIDS: Signal-aware explainable intrusion detection system for controller area network-based in-vehicle network. *IEEE Transactions on Vehicular Technology* 73:3230 – 3246, DOI 10.1109/TVT.2023.3327275, URL <https://ieeexplore.ieee.org/document/10294210>
44. Kaiser C, Stocker A, Festl A (2019) Automotive CAN bus data: An example dataset from the AEGIS big data project. Zenodo DOI 10.5281/zenodo.3267184, URL <https://zenodo.org/records/3267184>
45. Kang H, Kwak BI, Lee YH, Lee H, Lee H, Kim HK (2021) Car hacking and defense competition on in-vehicle network. Workshop on Automotive and Autonomous Vehicle Security (AutoSec) 2021 DOI 10.14722/autosec.2021.23035, URL https://www.ndss-symposium.org/wp-content/uploads/autosec2021_23035_paper.pdf
46. Kang H, Kwak BI, Lee YH, Lee H, Lee H, Kim HK (2021) Car hacking: Attack & defense challenge 2020. Hacking and Countermeasure Research Lab URL <https://ocslab.hksecurity.net/Datasets/carchallenge2020>
47. Kang H, Kwak BI, Lee YH, Lee H, Lee H, Kim HK (2021) Car hacking: Attack & defense challenge 2020 dataset. *IEEE Dataport* DOI 10.21227/qvr7-n418, URL <https://ieee-dataport.org/open-access/car-hacking-attack-defense-challenge-2020-dataset>
48. Kang MJ, Kang JW (2016) CAN packets. figshare DOI 10.1371/journal.pone.0155781.s001, URL https://figshare.com/articles/dataset/Intrusion_Detection_System_Using_Deep_Neural_Network_for_In-Vehicle_Network_Security/3425357?file=5360381
49. Kang MJ, Kang JW (2016) Intrusion detection system using deep neural network for in-vehicle network security. *PLOS One* DOI 10.1371/journal.pone.0155781, URL <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0155781>
50. Karopoulos G, Kambourakis G, Chatzoglou E, Hernández-Ramos JL, Kouliaridis V (2022) Demystifying in-vehicle intrusion detection systems: A survey of surveys and a meta-taxonomy. *Electronics* 11:1072, DOI 10.3390/electronics11071072, URL <https://www.mdpi.com/2079-9292/11/7/1072>
51. Keen Security Lab of Tencent (2016) Car hacking research: Remote attack tesla motors. Keen Security Lab Blog URL <https://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/>
52. Kidmose B (2024) A review of smart vehicles in smart cities: Dangers, impacts, and the threat landscape. *Vehicular Communications* 51:100,871, DOI 10.1016/j.vehcom.2024.100871, URL <https://www.sciencedirect.com/science/article/pii/S2214209624001463>
53. Kidmose B, Meng W (2024) can-fp: An attack-aware analysis of false alarms in automotive intrusion detection models. 21st Annual International Conference on Privacy, Security, and Trust (PST 2024)
54. Kidmose B, Meng W (2024) can-sleuth: Investigating and evaluating automotive intrusion detection datasets. 2024 European Interdisciplinary Cybersecurity Conference (EICC 2024) DOI 10.1145/3655693.3655696, URL <https://dl.acm.org/doi/10.1145/3655693.3655696>
55. Kim HK (2022) B-can intrusion dataset. Hacking and Countermeasure Research Lab URL <https://ocslab.hksecurity.net/Datasets/b-can-intrusion-dataset>
56. Kim HK (2022) Can-fd intrusion dataset. Hacking and Countermeasure Research Lab URL <https://ocslab.hksecurity.net/Datasets/can-fd-intrusion-dataset>
57. Kim HK (2022) M-can intrusion dataset. Hacking and Countermeasure Research Lab URL <https://ocslab.hksecurity.net/Datasets/m-can-intrusion-dataset>
58. Koscher K, Czeskis A, Roesner F, Shwetak Patel TK, Checkoway S, McCoy D, Kantor B, Anderson D, Shacham H, Savage S (2010) Experimental security analysis of a modern automobile. 2010 IEEE Symposium on Security and Privacy DOI 10.1109/SP.2010.34, URL <https://ieeexplore.>

- ieee.org/document/5504804
59. Lab C (2017) Previous dataset. Vehicle Security Research URL <https://www.crysys.hu/research/vehicle-security/>
 60. Lampe B (2023) can-train-and-test. Bitbucket URL <https://bitbucket.org/brooke-lampe/can-train-and-test/src/master/>
 61. Lampe B (2023) can-train-and-test-v1.5. Bitbucket URL <https://bitbucket.org/brooke-lampe/can-train-and-test-v1.5/src/master/>
 62. Lampe B, Meng W (2022) IDS for CAN: A practical intrusion detection system for CAN bus security. 2022 IEEE Global Communications Conference (GLOBECOM 2022) DOI 10.1109/GLOBECOM48099.2022.10001536, URL <https://ieeexplore.ieee.org/document/10001536>
 63. Lampe B, Meng W (2023) can-logic: Automotive intrusion detection via temporal logic. 13th International Conference on the Internet of Things (IoT 2023) pp 113 – 120, DOI 10.1145/3627050.3627059, URL <https://dl.acm.org/doi/10.1145/3627050.3627059>
 64. Lampe B, Meng W (2023) can-train-and-test: A new CAN intrusion detection dataset. 2023 IEEE 98th Vehicular Technology Conference (VTC 2023-Fall) DOI 10.1109/VTC2023-Fall60731.2023.10333756, URL <https://ieeexplore.ieee.org/document/10333756>
 65. Lampe B, Meng W (2023) Intrusion detection in the automotive domain: A comprehensive review. IEEE Communications Surveys & Tutorials 25:2356 – 2426, DOI 10.1109/COMST.2023.3309864, URL <https://ieeexplore.ieee.org/document/10233928>
 66. Lampe B, Meng W (2023) A survey of deep learning-based intrusion detection in automotive applications. Expert Systems with Applications 221:119,771, DOI 10.1016/j.eswa.2023.119771, URL <https://www.sciencedirect.com/science/article/pii/S0957417423002725>
 67. Lampe B, Meng W (2024) can-train-and-test: A curated CAN dataset for automotive intrusion detection. Computers & Security 140:103,777, DOI 10.1016/j.cose.2024.103777, URL <https://www.sciencedirect.com/science/article/pii/S0167404824000786>
 68. Lampe BE (2023) can-train-and-test. DTU Data DOI 10.11583/DTU.24805533, URL <https://data.dtu.dk/articles/dataset/can-train-and-test/24805533>
 69. Lee H, Jeong SH, Kim HK (2017) CAN dataset for intrusion detection (OTIDS). Hacking and Countermeasure Research Lab URL <https://ocslab.hksecurity.net/Dataset/CAN-intrusion-dataset>
 70. Lee H, Jeong SH, Kim HK (2017) OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame. 2017 15th Annual Conference on Privacy, Security and Trust (PST) DOI 10.1109/PST.2017.00017, URL <https://ieeexplore.ieee.org/document/8476919>
 71. Lee S, Jo HJ, Cho A, Lee DH, Choi W (2022) Survey to download the dataset [ttids]. Google Forms URL <https://forms.gle/FRL5Ptrzqh7DjJey9>
 72. Lee S, Jo HJ, Cho A, Lee DH, Choi W (2022) TTIDS: Transmission-resuming time-based intrusion detection system for controller area network (can). IEEE Access 10:52,139 – 52,153, DOI 10.1109/ACCESS.2022.3174356, URL <https://ieeexplore.ieee.org/document/9772644>
 73. Lee S, Choi W, Kim I, Lee G, Lee DH (2023) A comprehensive analysis of datasets for automotive intrusion detection systems. Computers, Materials & Continua 76(3):3413 – 3442, DOI 10.32604/cmc.2023.039583, URL <https://www.techscience.com/cmc/v76n3/54350>
 74. Lee S, Jo HJ, Cho A, Lee DH, Choi W (2023) Dataset. GitHub URL <https://github.com/EmbeddedSecurity/AutomotiveSecurity/tree/main/Dataset>
 75. Lemke K, Paar C, Wolf M (2006) Embedded Security in Cars. Springer-Verlag, URL <https://link.springer.com/book/10.1007/3-540-28428-1>
 76. Lokman SF, Othman AT, Abu-Bakar MH (2019) Intrusion detection system for automotive controller area network (can) bus system: a review. EURASIP Journal on Wireless Communications and Networking 2019, URL <https://jwcn-urasipjournals.springeropen.com/articles/10.1186/s13638-019-1484-3>
 77. Lu R, Lin X, Zhu H, Ho PH, Shen X (2008) Ecpp: Efficient conditional privacy preservation protocol for secure vehicular communications. IEEE INFOCOM 2008 - The 27th Conference on Computer Communications URL <https://ieeexplore.ieee.org/document/4509774>
 78. López OAM, López AM, Crossa J (2022) Overfitting, Model Tuning, and Evaluation of Prediction Performance., Springer International Publishing, p 109 – 139. DOI 10.1007/978-3-030-89010-0_4, URL https://link.springer.com/chapter/10.1007/978-3-030-89010-0_4
 79. Marchetti M, Stabili D (2017) Anomaly detection of can bus messages through analysis of id sequences. 2017 IEEE Intelligent Vehicles Symposium (IV) pp 1577 – 1583, URL <https://ieeexplore.ieee.org/document/7995934>

80. Michael SS (2019) Introduction to can (controller area network). All About Circuits URL <https://www.allaboutcircuits.com/technical-articles/introduction-to-can-controller-area-network/>
81. Miller C (2019) Lessons learned from hacking a car. IEEE Design & Test 36:7 – 9, DOI 10.1109/MDAT.2018.2863106, URL <https://ieeexplore.ieee.org/document/8890036>
82. Miller C, Valasek C (2015) Remote exploitation of an unaltered passenger vehicle. IOActive URL https://ioactive.com/wp-content/uploads/2018/05/IOActive_Remote_Car_Hacking-1.pdf
83. Miller C, Valasek C (2015) Remote exploitation of an unaltered passenger vehicle. BlackHat URL <https://archive.org/details/youtube-MaChkASmXEc>
84. Miller C, Valasek C (2016) Advanced can injection techniques for vehicle networks. BlackHat URL <https://archive.org/details/youtube-4wgEmNlu20c>
85. Miller C, Valasek C (2016) Can message injection. Illmatics URL <https://illmatics.com/can%20message%20injection.pdf>
86. MLatETAS, Wirth G (2019) SynCAN. GitHub URL <https://github.com/etas/SynCAN/tree/master>
87. Mokhadder M, Zachos M, Potter J (2023) Evaluation of vehicle system performance of an sae j1939-91c network security implementation. In: WCX SAE World Congress Experience, SAE International, DOI 10.4271/2023-01-0041, URL <https://saemobilus.sae.org/content/2023-01-0041>
88. Moustafa N, Slay J (2015) Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). 2015 Military Communications and Information Systems Conference (MilCIS) DOI 10.1109/MilCIS.2015.7348942, URL <https://ieeexplore.ieee.org/document/7348942>
89. Moustafa N, Slay J (2015) The unsw-nb15 dataset. University of New South Wales (UNSW), Sydney URL <https://research.unsw.edu.au/projects/unsw-nb15-dataset>
90. Mouzakitis OYAJCMMDDOA (2019) Intrusion detection systems for intra-vehicle networks: A review. IEEE Access 7:21,266 – 21,289, DOI 10.1109/ACCESS.2019.2894183, URL <https://ieeexplore.ieee.org/document/8642311>
91. Nie S, Liu L, Du Y (2017) Free-fall: Hacking tesla from wireless to can bus. BlackHat URL <https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf>
92. nkmk (2023) Maximum and minimum float values in python. notenkkmkme URL <https://note.nkmk.me/en/python-sys-float-info-max-min/>
93. Ocheri V, Sheng Z, Ali F (2017) A survey of automotive networking applications and protocols. In: Connected Vehicle Systems, CRC Press, URL <https://www.taylorfrancis.com/chapters/edit/10.4324/9781315232928-1/survey-automotive-networking-applications-protocols-victor-ocheri-zhengguo-sheng-falah-ali>
94. pandas Contributors (2023) pandas. pandas URL <https://pandas.pydata.org/>
95. Parekh N, Campau T (2024) Average age of vehicles hits new record in 2024. S&P Global Mobility URL <https://www.spglobal.com/mobility/en/research-analysis/average-age-vehicles-united-states-2024.html>
96. 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, Édouard Duchesnay (2011) Scikit-learn: Machine learning in python. Journal of Machine Learning Research 12:2825 – 2830, URL <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>
97. Pollicino F, Stabili D, Marchetti M (2024) Material used for the submission at ACM transactions on cyber-physical systems - special issue on automotive CPS safety & security. Web Engineering and Benchmarking Laboratory URL <https://weblab.ing.unimore.it/people/stabili/resources/tcps.shtml>
98. Pollicino F, Stabili D, Marchetti M (2024) Performance comparison of timing-based anomaly detectors for controller area network: A reproducible study. ACM Transactions on Cyber-Physical Systems 8:1 – 24, DOI 10.1145/360491, URL <https://dl.acm.org/doi/10.1145/3604913>
99. Popa L, Groza B, Jichici C, Murvay PS (2022) ECUPrint - physical fingerprinting electronic control units on CAN buses inside cars and SAE j1939 compliant vehicles. GitHub URL <https://github.com/LucianPopaLP/ECUPrint>
100. Popa L, Groza B, Jichici C, Murvay PS (2022) ECUPrint—physical fingerprinting electronic control units on CAN buses inside cars and SAE j1939 compliant vehicles. IEEE Transactions on Information Forensics and Security 17:1185 – 1200, DOI 10.1109/TIFS.2022.3158055, URL <https://ieeexplore.ieee.org/document/9730883>
101. Poulsen K (2010) Hacker disables more than 100 cars remotely. Wired URL <https://www.wired.com>

- com/2010/03/hacker-bricks-cars/
102. Quinton S, Bone TT, Hennig J, Neukirchner M, Negrean M, Ernst R (2014) Typical worst case response-time analysis and its use in automotive network design. DAC '14: Proceedings of the 51st Annual Design Automation Conference DOI 10.1145/2593069.260297, URL <https://dl.acm.org/doi/10.1145/2593069.260297>
 103. Rajapaksha S, Kalutarage H, Al-Kadri MO, Petrovski A, Madzudzo G, Cheah M (2023) Ai-based intrusion detection systems for in-vehicle networks: A survey. ACM Computing Surveys 55:1 – 40, DOI 10.1145/3570954, URL <https://dl.acm.org/doi/10.1145/3570954>
 104. Rajapaksha S, Kalutarage H, Madzudzo G, Petrovski A, Al-Kadri M (2024) Can-mirgu. GitHub URL <https://github.com/sampathraajaksha/CAN-MIRGU>
 105. Rajapaksha S, Kalutarage H, Madzudzo G, Petrovski A, Al-Kadri M (2024) Can-mirgu. Google Drive URL https://drive.google.com/drive/folders/1uUKLEu_tFVMY9WkDnf1rqQPwuQLQFwBL?usp=sharing
 106. Rajapaksha S, Kalutarage H, Madzudzo G, Petrovski A, Al-Kadri M (2024) CAN-MIRGU: A comprehensive CAN bus attack dataset from moving vehicles for intrusion detection system evaluation. Symposium on Vehicle Security and Privacy (VehicleSec) 2024 URL <https://www.ndss-symposium.org/wp-content/uploads/vehicsec2024-43-paper.pdf>
 107. Raya M, Hubaux JP (2007) Securing vehicular ad hoc networks. Journal of Computer Security 15:39 – 68, URL <https://content.iospress.com/articles/journal-of-computer-security/jcs275>
 108. Refat RUD, Elkhail AA, Malik H (2022) Machine Learning for Automotive Cybersecurity: Challenges, Opportunities and Future Directions., Springer International Publishing, p 547 – 567. URL https://link.springer.com/chapter/10.1007/978-3-031-06780-8_20
 109. Robert Bosch GmbH (2022) Comparison of classical can, can fd, and can xl and can xl and ethernet 10base-t1s. Automotive Electronics (AE/PAI-IP) URL https://www.bosch-semiconductors.com/media/ip_modules/pdf_2/can_xl_1/20220825_can_xl_vs_10base-t1s_v2.pdf
 110. Roque ADS (2024) CAN-Modes-datasets. GitHub URL <https://github.com/Asr-roque/canmodes-datasets>
 111. Roque ADS (2025) Can-modes: In-vehicle datasets generation and analysis in different driving situations. IEEE Dataport DOI 10.21227/j1ts-bm15, URL <https://ieee-dataport.org/documents/can-modes-vehicle-datasets-different-driving-situations>
 112. Roque ADS, Alves LMDS, de Freitas EP (2024) Can-modes: In-vehicle datasets generation and analysis in different driving situations. 2024 Workshop on Communication Networks and Power Systems (WCNPS) DOI 10.1109/WCNPS65035.2024.10814379, URL <https://ieeexplore.ieee.org/document/10814379>
 113. Sami M (2019) Intrusion detection in can bus. IEEE Dataport DOI 10.21227/24m9-a446, URL <https://dx.doi.org/10.21227/24m9-a446>
 114. Sami M, Ibarra M, Esparza AC, 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. 2020 IEEE Green Energy and Smart Systems Conference (IGESSC) DOI 10.1109/IGESSC50231.2020.9285088, URL <https://ieeexplore.ieee.org/document/9285088>
 115. Schüßler J (2016) Erpressungstrojaner “highwayman” zielt auf autofahrer. Heise Medien
 116. scikit-learn developers (2024) Developing scikit-learn estimators (1.5.1). scikit-learn: Machine Learning in Python URL <https://scikit-learn.org/stable/developers/develop.html>
 117. scikit-learn developers (2025) accuracy_score. scikit-learn: Machine Learning in Python URL https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
 118. scikit-learn developers (2025) f1_score. scikit-learn: Machine Learning in Python URL https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score
 119. scikit-learn developers (2025) precision_score. scikit-learn: Machine Learning in Python URL https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
 120. scikit-learn developers (2025) recall_score. scikit-learn: Machine Learning in Python URL https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score
 121. Seeam A, Ogbeh OS, Guness S, Bellekens X (2019) Threat modeling and security issues for the internet of things. 2019 Conference on Next Generation Computing Applications (NextComp) URL <https://ieeexplore.ieee.org/document/8883642>

122. Seo E, Song HM, Kim HK (2018) Car-hacking dataset for the intrusion detection. Hacking and Countermeasure Research Lab URL <https://ocslab.hksecurity.net/Datasets/car-hacking-dataset>
123. Seo E, Song HM, Kim HK (2018) GIDS: GAN based intrusion detection system for in-vehicle network. 2018 16th Annual Conference on Privacy, Security and Trust (PST) DOI 10.1109/PST.2018.8514157, URL <https://ieeexplore.ieee.org/document/8514157>
124. Shakarian P, Shakarian J, Ruef A (2013) Chapter 13 - attacking iranian nuclear facilities: Stuxnet. In: Introduction to Cyber-Warfare, Syngress, pp 223 – 239, DOI 10.1016/B978-0-12-407814-7.00013-0, URL <https://www.sciencedirect.com/science/article/pii/B9780124078147000130>
125. Sharmin S, Mansor H, Kadir AFA, Aziz NA (2024) Benchmarking frameworks and comparative studies of controller area network (can) intrusion detection systems: A review. Journal of Computer Security 32, DOI 10.3233/JCS-230027, URL <https://journals.sagepub.com/doi/full/10.3233/JCS-230027>
126. Shreejith S, Mundhenk P, Ettner A, Fahmy SA, Steinhorst S, Lukasiewicz M, Chakraborty S (2017) VEGA: A high performance vehicular ethernet gateway on hybrid FPGA. IEEE Transactions on Computers 66:1790 – 1803, DOI 10.1109/TC.2017.2700277, URL <https://ieeexplore.ieee.org/document/7917319>
127. Siddiqui AS, Gui Y, Plusquellic J, Saqib F (2017) Secure communication over canbus. 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS) URL <https://ieeexplore.ieee.org/document/8053160>
128. Song HM, Kim HK (2020) CAN signal extraction and translation dataset. Hacking and Countermeasure Research Lab URL <https://ocslab.hksecurity.net/Datasets/can-signal-extraction-and-translation-dataset>
129. Song HM, Kim HK (2020) Discovering CAN specification using on-board diagnostics. IEEE Design & Test 38:93 – 103, DOI 10.1109/MDAT.2020.3011036, URL <https://ieeexplore.ieee.org/document/9145748>
130. Song HM, Woo J, Kim HK (2019) In-vehicle network intrusion detection using deep convolutional neural network. Vehicular Communications 21:100,198, DOI 10.1016/j.vehcom.2019.100198, URL <https://www.sciencedirect.com/science/article/pii/S2214209619302451>
131. Stabili D, Marchetti M (2019) Detection of missing CAN messages through inter-arrival time analysis. 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall) DOI 10.1109/VTCFall.2019.8891068, URL <https://ieeexplore.ieee.org/document/8891068>
132. Stabili D, Marchetti M (2019) Resources for automotive cyber-security research: Bus-off dataset. Web Engineering and Benchmarking Laboratory URL <https://weblab.ing.unimore.it/people/stabili/resources/>
133. Stabili D, Ferretti L, Andreolini M, Marchetti M (2022) DAGA: Detecting attacks to in-vehicle networks via n-gram analysis. IEEE Transactions on Vehicular Technology 71:11,540 – 11,554, DOI 10.1109/TVT.2022.3190721, URL <https://ieeexplore.ieee.org/document/9829329>
134. Stabili D, Ferretti L, Andreolini M, Marchetti M (2022) Resources for automotive cyber-security research: DAGA dataset. Web Engineering and Benchmarking Laboratory URL <https://weblab.ing.unimore.it/people/stabili/resources/>
135. Swessi D, Idoudi H (2021) A comparative review of security threats datasets for vehicular networks. 2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT) DOI 10.1109/3ICT53449.2021.9581683, URL <https://ieeexplore.ieee.org/document/9581683>
136. Szydlowski CP (1992) Can specification 2.0: Protocol and implementations. SAE Technical Paper DOI 10.4271/921603, URL <https://www.sae.org/publications/technical-papers/content/921603/>
137. Tavallae M, Bagheri E, Lu W, Ghorbani AA (2009) A detailed analysis of the kdd cup 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications DOI 10.1109/CISDA.2009.5356528, URL <https://ieeexplore.ieee.org/document/5356528>
138. Tavallae M, Bagheri E, Lu W, Ghorbani AA (2009) Iscx nsl-kdd dataset 2009. University of New Brunswick URL <https://www.unb.ca/cic/datasets/nsl.html>
139. Tindell K (2023) Can injection: keyless car theft. Canis Automotive Labs URL <https://kentindell.github.io/2023/04/03/can-injection/>
140. Ueda H, Kurachi R, Takada H, Mizutani T, Inoue M, Horihata S (2015) Security authentication system for in-vehicle network. Sei Technical Review URL <https://global-sei.com/technology/tr/bn81/pdf/81-01.pdf>

141. United States Department of Transportation: Bureau of Transportation Statistics (2024) Average age of automobiles and trucks in operation in the united states. National Transportation Statistics URL <https://www.bts.gov/content/average-age-automobiles-and-trucks-operation-united-states>
142. University of Turku (2021) CAN bus dataset collected from a heavy-duty truck. Fairdata DOI 10.23729/3160254e-85e9-4268-a636-5b3e54091706, URL <https://etsin.fairdata.fi/dataset/7586f24f-c91b-41df-92af-283524de8b3e>
143. Vahidi A, Rosenstatter T, Mowla NI (2022) Systematic evaluation of automotive intrusion detection datasets. Proceedings of the 6th ACM Computer Science in Cars Symposium (CSCS '22) DOI 10.1145/3568160.3570226, URL <https://dl.acm.org/doi/10.1145/3568160.3570226>
144. Verendel V, Nilsson DK, Larson UE, Jonsson E (2008) An approach to using honeypots in in-vehicle networks. 2008 IEEE 68th Vehicular Technology Conference URL <https://ieeexplore.ieee.org/document/4657092>
145. Verma ME, Bridges RA, Iannacone MD, Hollifield SC, Moriano P, Hespeler SC, Kay B, Combs FL (2020) Real ORNL automotive dynamometer (ROAD) CAN intrusion dataset. Zenodo DOI 10.5281/zenodo.10462796, URL <https://zenodo.org/records/10462796>
146. Verma ME, Bridges RA, Iannacone MD, Hollifield SC, Moriano P, Hespeler SC, Kay B, Combs FL (2024) A comprehensive guide to CAN IDS data and introduction of the ROAD dataset. PLOS One DOI 10.1371/journal.pone.0296879, URL <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0296879>
147. Voss W (2019) SAE j1939 bandwidth, busload and message frame frequency. Copperhill Technologies: Blog URL <https://copperhilltech.com/blog/sae-j1939-bandwidth-busload-and-message-frame-frequency/>
148. Wang D, Ganesan S (2021) Automotive network security. 2021 IEEE International Conference on Electro Information Technology (EIT) URL <https://ieeexplore.ieee.org/document/9491889>
149. Wang Q, Sawhney S (2014) Vecure: A practical security framework to protect the can bus of vehicles. 2014 International Conference on the Internet of Things (IOT) URL <https://ieeexplore.ieee.org/document/7030108>
150. Wolf M, Lambert R (2017) Hacking Trucks - Cybersecurity Risks and Effective Cybersecurity Protection for Heavy Duty Vehicles., Gesellschaft für Informatik, Bonn, pp 45 – 60. Lecture Notes in Informatics (LNI), URL <https://dl.gi.de/items/5dd03474-71ca-4c58-97ed-b2be501b1238>
151. Woo S, Jo HJ, Lee DH (2014) A practical wireless attack on the connected car and security protocol for in-vehicle can. IEEE Transactions on Intelligent Transportation Systems 16:993 – 1006, DOI 10.1109/TITS.2014.2351612, URL <https://ieeexplore.ieee.org/document/6894181>
152. 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 Transactions on Intelligent Transportation Systems 21:919 – 933, DOI 10.1109/TITS.2019.2908074, URL <https://ieeexplore.ieee.org/document/8688625>
153. Yang L, Moubayed A, Hamieh I, Shami A (2019) Tree-based intelligent intrusion detection system in internet of vehicles. 2019 IEEE Global Communications Conference (GLOBECOM) URL <https://ieeexplore.ieee.org/document/9013892>
154. Zago M, Longari S, Tricarico A, Carminati M, Pérez MG, Pérez GM, Zanero S (2020) ReCAN data - reverse engineering of controller area networks. Mendeley Data DOI 10.17632/76knkx3fzv.2, URL <https://data.mendeley.com/datasets/76knkx3fzv/2>
155. Zago M, Longari S, Tricarico A, Carminati M, Pérez MG, Pérez GM, Zanero S (2020) ReCAN – dataset for reverse engineering of controller area networks. Data in Brief 29:105,149, DOI 10.1016/j.dib.2020.105149, URL <https://www.sciencedirect.com/science/article/pii/S2352340920300433>
156. Zhang Y, Liu T, Zhao H, Ma C (2021) Risk analysis of can bus and ethernet communication security for intelligent connected vehicles. 2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID) URL <https://ieeexplore.ieee.org/document/9491889>
157. Zheng B, Li W, Deng P, Gérard L, Zhu Q, Shankar N (2015) Design and verification for transportation system security. DAC '15: Proceedings of the 52nd Annual Design Automation Conference pp 1 – 6, URL <https://dl.acm.org/doi/10.1145/2744769.2747920>

A Extended CAN Traffic Captures

Listing 5 presents attack-free CAN traffic collected from a 2011 Chevrolet Traverse, while Listing 6 shows attack-free CAN traffic from a 2017 Subaru Forester. As discussed in Section 7.2, CAN traffic can vary significantly between different vehicles. A comparison of these two listings reveals that none of the arbitration IDs in Listing 5 overlap with those in Listing 6, highlighting the distinct differences in CAN traffic across different vehicle models.

Listing 5 Attack-free CAN traffic captured from a 2011 Chevrolet Traverse.

```
(1672163924.621479) 0C1#12E55F82124B979F
(1672163924.621724) 0C5#11D90EB31221AB2D
(1672163924.621942) 1E5#46FFD8C000002600
(1672163924.622330) 1F3#C0E0
(1672163924.624005) 0C7#02629A3D
(1672163924.624251) 0F9#07CD400000B828FF
(1672163924.624469) 189#0FFF0FFF300128FF
(1672163924.624722) 199#0FFF0E70F19000FF
(1672163924.628004) 1CB#100000
(1672163924.629932) 0C9#8409530D00010000
(1672163924.630125) 191#06C406D506C50000
(1672163924.630306) 0F1#224400C0
(1672163924.630530) 1ED#4133012201240800
(1672163924.630703) 1EF#0000019B
(1672163924.630947) 1A1#0010414000000000
(1672163924.631251) 1C3#06C406C500000000
(1672163924.631348) 1EB#0136
(1672163924.631774) 0C1#22E55F82224B979F
(1672163924.632014) 2C3#081706C606C64400
(1672163924.632250) 0C5#21D90EB32221AB2D
(1672163924.632447) 184#0001002001DF
(1672163924.632695) 19D#80003FFE000000FF
(1672163924.632914) 1C7#0FFF700003FF3F
(1672163924.633065) 1AF#000000
(1672163924.633251) 1CD#C7FF07FE7F
(1672163924.633448) 1E1#00FE120040
(1672163924.633667) 1E5#46FFD8E000002500
(1672163924.633918) 1E9#4003000C00000000
(1672163924.634167) 1F5#0101000400000900
(1672163924.634301) 334#0000
(1672163924.636531) 0C7#02629A3D
(1672163924.636780) 0F9#07CD400000B828FF
(1672163924.637055) 189#4FFF0FFF300028FF
(1672163924.637250) 199#4FFF0E70F18F00FF
(1672163924.639528) 0F1#3E4400C0
(1672163924.639932) 0C9#8409530000010000
(1672163924.640153) 191#06C406D506C50000
(1672163924.640382) 1ED#4133012201250800
(1672163924.640551) 1EF#0000019B
(1672163924.640802) 3C1#0765E70000000000
(1672163924.641050) 3D1#0116000000370000
(1672163924.641301) 3E9#00001015000010AB
(1672163924.641521) 0C1#32E55F82324B979F
(1672163924.641747) 0C5#31D90EB33221AB2D
(1672163924.641995) 1E5#46FFD78F9C002900
(1672163924.642168) 2F9#A801210000
(1672163924.642348) 348#00000000
(1672163924.642525) 34A#00000000
(1672163924.643455) 1EB#0128
(1672163924.648010) 1CB#100000
```

Listing 6 Attack-free CAN traffic captured from a 2017 Subaru Forester.

```
(1672865121.432848) 002#07317005AD000000
(1672865121.434040) 075#000000000009033B
(1672865121.434295) 0D0#E3F81700010400F5
(1672865121.434467) 0D1#5C000080
(1672865121.434720) 0D2#0000FFFF00010000
(1672865121.434935) 0D3#0005C00F602363
(1672865121.435172) 070#EF8400FFAB7F039A
(1672865121.435422) 080#1B800000427C033E
(1672865121.435668) 0D4#5700620054005F00
(1672865121.435878) 140#1401450500141AA0
(1672865121.436172) 141#EA28DC2945452002
(1672865121.436397) 144#00006D332EA80040
(1672865121.436495) 150#0000
(1672865121.436728) 360#5E046C7120400000
(1672865121.436985) 361#002A00D001000000
(1672865121.437219) 362#F37B6FC85243E014
(1672865121.437442) 576#00000000162523A3
(1672865121.437680) 152#F16400000000008C
(1672865121.437813) 376#FA
(1672865121.438035) 6FC#0116042801054A00
(1672865121.438294) 148#0B1B1F8241001105
(1672865121.438533) 156#018000000000000F
(1672865121.438764) 149#7D013FABAB430000
(1672865121.438994) 371#680096C7E678E020
(1672865121.442848) 002#07357007B3000000
(1672865121.443085) 280#0800100000000000
(1672865121.445743) 140#1402450500141AA0
(1672865121.445967) 141#E728DC2945052002
(1672865121.448044) 148#0B1C238241001105
(1672865121.448265) 149#7D013FABAB430000
(1672865121.449800) 160#0000000000200323
(1672865121.450016) 161#282386FF00300000
(1672865121.450269) 162#00001F0224000300
(1672865121.450508) 163#0000400000003000F
(1672865121.450771) 164#0100000000000001
(1672865121.451018) 22A#D123000030FF7F00
(1672865121.452848) 372#0008000000000000
(1672865121.453059) 002#073A7009BA000000
(1672865121.454173) 075#0000000000090468
(1672865121.454341) 0D0#DAF81700010400F4
(1672865121.454545) 0D1#5D000000
(1672865121.454756) 0D2#0000FFFF00010000
(1672865121.455009) 070#FB8400FFB97F0402
(1672865121.455244) 080#17800000237C0409
(1672865121.455467) 0D3#8005C00F602364
(1672865121.455706) 0D4#5700630055006000
(1672865121.455938) 140#14035B0500141A20
(1672865121.456169) 141#E728DC2945052002
(1672865121.456416) 144#00006D332EA80040
(1672865121.456531) 150#0000
```

B Additional Results

Each of our machine learning models was configured with default parameters (the `timestamp` feature was included, the `data field` feature was subdivided) and pitted against `can-train-and-test-v1.5`. Each model was evaluated against all four sub-datasets and all six testing subsets. Even the suppress attacks are included. The results are shown in Table 29.

Table 30 presents the averaged results of experiments conducted on both the HCRL Car Hacking dataset and the HCRL Survival Analysis dataset. By aggregating these two datasets, we provide a generalized perspective that can be compared against the `can-train-and-test-v1.5` dataset. While this aggregated view offers an interesting data point, it should be considered alongside the individual dataset results found in Tables 21 and 22 for a more nuanced understanding.

Table 31 provides a comprehensive average of experiments across the three UNIMORE datasets (`bus-off`, `daga`, and `ventus`), including all sub-datasets and all testing subsets. Similar to Table 29, these averages incorporate “removal” (also known as “suppress”) attacks, which should be interpreted with caution due to the lack of labeled data, which precludes a meaningful evaluation.

Four machine learning models performed better against `can-train-and-test-v1.5` than `can-unimore-curated-v3`; twelve performed worse. That said, the suppress attacks were assessed using metrics that depend on labeled data—suppress attacks suppress (or “remove”) data; as such, they cannot be labeled. Thus, the results of experiments involving suppress attacks are uninformative, and since Tables 29 and 31 contain the suppress attacks, they can also be misleading. As such, we moved Tables 29 and 31 to the appendix as interesting—but not particularly insightful—pieces of data.

Similarly, we include Table 32 in the appendix; Table 32 contains the average results of *all* the `ventus` experiments—both the “injection” experiments and the “removal” (i.e., suppress) experiments. This table complements Table 26, which excludes the uninformative “removal” attack.

Table 29: can-train-and-test-v1.5: Averages with default parameters; *include* testing subsets #5 and #6.¹

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9517	0.9822	0.9517	0.9654	0.9560	0.0396
K-Nearest Neighbors	0.9404	0.9858	0.9404	0.9594	0.9439	0.0526
Linear Regression	0.3410	0.9842	0.3410	0.4809	0.3384	0.6641
Logistic Regression	0.9873	0.9855	0.9873	0.9850	0.9916	0.0040
Linear Support Vector Machine	0.9886	0.9821	0.9886	0.9852	0.9932	0.0021
Decision Tree	0.6922	0.9878	0.6922	0.7900	0.6927	0.3068
Extra Trees	0.8872	0.9864	0.8872	0.9256	0.8902	0.1067
Gradient Boosting	0.7793	0.9866	0.7793	0.8554	0.7810	0.2173
Random Forest	0.8897	0.9868	0.8897	0.9301	0.8926	0.1044
Isolation Forest	0.5505	0.9805	0.5505	0.6987	0.5512	0.4480
K-Means	0.4122	0.9830	0.4122	0.5555	0.4100	0.5920
Mini-Batch K-Means	0.5204	0.9815	0.5204	0.6545	0.5212	0.4781
BIRCH	0.4808	0.9819	0.4808	0.5419	0.4815	0.5177
Local Outlier Factor	0.3403	0.9768	0.3403	0.4637	0.3393	0.6617
Multi-Layer Perceptron	0.9347	0.9866	0.9347	0.9572	0.9381	0.0585
Bernoulli Restricted Boltzmann Machine	0.9920	0.9842	0.9920	0.9881	0.9960	0.0000

¹ This evaluation was conducted to compare the can-train-and-test-v1.5 dataset with the UNIMORE Bus-Off (bus-off), DAGA (daga), and Ventus (ventus) datasets. Testing subsets #5 and #6 were averaged into the performance metrics in this table, since there are similar attack types in several of the UNIMORE datasets.

² test_05_suppress lacks labeled CAN frames and cannot be effectively evaluated. The table shows baseline IDS performance, not detection capability.

Table 30: hcr1-ch, hcr1-sa: Averages with default parameters.

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.7910	0.7545	0.7910	0.7571	0.8554	0.0742
K-Nearest Neighbors	0.8532	0.8750	0.8532	0.8544	0.8911	0.0690
Linear Regression	0.3668	0.8417	0.3668	0.3853	0.3088	0.7364
Logistic Regression	0.8293	0.7645	0.8293	0.7679	0.9025	0.0171
Linear Support Vector Machine	0.8339	0.7609	0.8339	0.7680	0.9082	0.0103
Decision Tree	0.6898	0.8506	0.6898	0.7285	0.6899	0.3079
Extra Trees	0.8529	0.8629	0.8529	0.8493	0.8890	0.0726
Gradient Boosting	0.9376	0.9356	0.9376	0.9340	0.9500	0.0371
Random Forest	0.8562	0.8933	0.8562	0.8655	0.8656	0.1243
Isolation Forest	0.4544	0.7783	0.4544	0.5135	0.4309	0.5892
K-Means	0.6110	0.7905	0.6110	0.6520	0.6157	0.3775
Mini-Batch K-Means	0.7597	0.7984	0.7597	0.7711	0.7866	0.1840
BIRCH	0.5667	0.7601	0.5667	0.5705	0.5839	0.3934
Local Outlier Factor	0.1856	0.6106	0.1856	0.1199	0.0773	0.9502
Multi-Layer Perceptron	0.8779	0.8894	0.8779	0.8681	0.9176	0.0402
Bernoulli Restricted Boltzmann Machine	0.8419	0.7108	0.8419	0.7703	0.9172	0.0000

Table 31: can-unimore-curated-v3, all: Averages with default parameters.

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9806	0.9859	0.9806	0.9831	0.9840	0.0125
K-Nearest Neighbors	0.9889	0.9882	0.9889	0.9885	0.9919	0.0051
Linear Regression	0.5330	0.9894	0.5330	0.6261	0.5302	0.4725
Logistic Regression	0.6809	0.9791	0.6809	0.7557	0.6829	0.3151
Linear Support Vector Machine	0.9872	0.9769	0.9872	0.9812	0.9935	0.0002
Decision Tree	0.9902	0.9889	0.9902	0.9895	0.9929	0.0043
Extra Trees	0.9894	0.9888	0.9894	0.9891	0.9922	0.0051
Gradient Boosting	0.9912	0.9837	0.9912	0.9874	0.9953	0.0005
Random Forest	0.9906	0.9889	0.9906	0.9897	0.9933	0.0039
Isolation Forest	0.2976	0.9855	0.2976	0.4496	0.2955	0.7066
K-Means	0.4977	0.9803	0.4977	0.6556	0.4988	0.5000
Mini-Batch K-Means	0.4934	0.9824	0.4934	0.6508	0.4935	0.5063
BIRCH	0.7532	0.9851	0.7532	0.8312	0.7551	0.2430
Local Outlier Factor	0.5731	0.9881	0.5731	0.7168	0.5724	0.4282
Multi-Layer Perceptron	0.9926	0.9886	0.9926	0.9905	0.9956	0.0014
Bernoulli Restricted Boltzmann Machine	0.9928	0.9859	0.9928	0.9893	0.9964	0.0000

Table 32: can-unimore-curated-v3, ventus: Averages with default parameters; *include* testing subset #2.¹

Model	Accuracy	Precision	Recall	F1-score	G-mean	FPR
Gaussian Naive Bayes	0.9516	0.9771	0.9516	0.9639	0.9570	0.0375
K-Nearest Neighbors	0.9787	0.9775	0.9787	0.9780	0.9843	0.0101
Linear Regression	0.3761	0.9864	0.3761	0.5328	0.3730	0.6302
Logistic Regression	0.3761	0.9864	0.3761	0.5328	0.3730	0.6302
Linear Support Vector Machine	0.9887	0.9819	0.9887	0.9838	0.9942	0.0003
Decision Tree	0.9799	0.9775	0.9799	0.9787	0.9855	0.0088
Extra Trees	0.9775	0.9774	0.9775	0.9774	0.9831	0.0113
Gradient Boosting	0.9882	0.9776	0.9882	0.9828	0.9938	0.0005
Random Forest	0.9800	0.9775	0.9800	0.9787	0.9856	0.0088
Isolation Forest	0.2940	0.9750	0.2940	0.4437	0.2921	0.7097
K-Means	0.4984	0.9751	0.4984	0.6536	0.4989	0.5005
Mini-Batch K-Means	0.4802	0.9742	0.4802	0.6355	0.4806	0.5190
BIRCH	0.6031	0.9849	0.6031	0.6972	0.6027	0.3975
Local Outlier Factor	0.5202	0.9819	0.5202	0.6727	0.5194	0.4813
Multi-Layer Perceptron	0.9848	0.9775	0.9848	0.9811	0.9904	0.0039
Bernoulli Restricted Boltzmann Machine	Out of memory	Out of memory	Out of memory	Out of memory	Out of memory	Out of memory

¹ test_02_removal lacks labeled CAN frames and cannot be effectively evaluated. The table shows baseline IDS performance, not detection capability.