Efficient Voltage-based Intrusion Detection for in-Vehicle Networks: from Density Clustering to Centroid Classification

Patricia Iosif[a], Lucian Popa[a] and Bogdan Groza[a,*]

[a]Faculty of Automatics and Computers, Politehnica University of Timisoara, Timisoara, Romania

ARTICLE INFO

Keywords:
Controller Area Networks, voltage fingerprinting, clustering, intrusion detection

ABSTRACT

Detecting intrusions on in-vehicle networks from voltage characteristics has become a popular technique. However, an effective mechanism for voltage identification of Electronic Control Units requires both a sound clustering algorithm to determine the correct number of devices on the network and an efficient classifier that allows updates in order to handle changes due to environmental conditions. Firstly, we explore the use of HDBSCAN in order to cluster ECUs based on voltage characteristics. While HDBSCAN is a highly effective algorithm, which has the merit of having only a few parameters that need to be tuned, our results show that finding the optimal parametrization is not that straight-forward. We test two well-known methods and an empirical selection in order to determine optimal choices for the largest existing dataset that contains voltage samples from ten vehicles. Secondly, we use the Nearest Centroid classifier to identify ECUs based on their fingerprints, which offers the advantage of an extremely small memory footprint and an efficient updating mechanism for the centroids. Thus, the method is both efficient and capable of adapting to environmental changes, which is a known demand for voltage-based identification. The proposed methodology demonstrates a very high detection rate that is specific to voltage-based techniques, i.e., true acceptance rate greater than 99.93% and false acceptance rate lower than 0.03%, even when faced with changing environmental conditions when updates are used. It also features an easy to update mechanism and a minimal memory footprint that is 4 to 20 times smaller than baseline classifiers such as SVM and RF.

1. Introduction and Motivation

Modern vehicles are far from being a well-secured environment. Recent attacks have exploited vulnerabilities such as memory corruption inside Tesla's built-in browser [1] or mandatory Electronic Logging Devices (ELDs) installed inside trucks [2] – all these happened while the vehicle was on the road. When stationary, recent research has shown that through Electromagnetic Interference (EMI) attacks on smart glasses left inside a locked vehicle, attackers can activate the built-in voice assistant and, through pre-defined voice commands, potentially unlock and gain access to the vehicle [3]. While the attack surfaces can be spectacular, at the heart of many in-vehicle functionalities, a bus designed in the 80s still stays: the Controller Area Network (CAN) [4]. It facilitates the exchange of messages between Electronic Control Units (ECUs) responsible for various in-vehicle functions and thus, securing it is still a critical task.

Detecting changes in CAN bus topology and identifying the source of potentially adversarial messages can help design robust intrusion detection systems (IDS) to mitigate such attacks. In this context, using voltage characteristics is a popular technique for identifying in-vehicle ECUs and topology changes. It is indeed more demanding in terms of signal acquisition costs and processing demands, but given the safety-critical nature of in-vehicle functions, achieving high confidence in ECU identification is preferable. While most works focus on identifying in-vehicle ECUs, there are only a few that focus explicitly on ECU clustering. The main reason for this is that most works are pursuing the application of voltage-based recognition within the scope of intrusion detection. On the other hand, clustering in-vehicle ECUs can serve as mapping for recognizing how many devices are inside the vehicle which is valuable both for understanding if the network was altered or as a digital fingerprint of the car [5]. The largest voltage-based dataset, ECUPrint, was

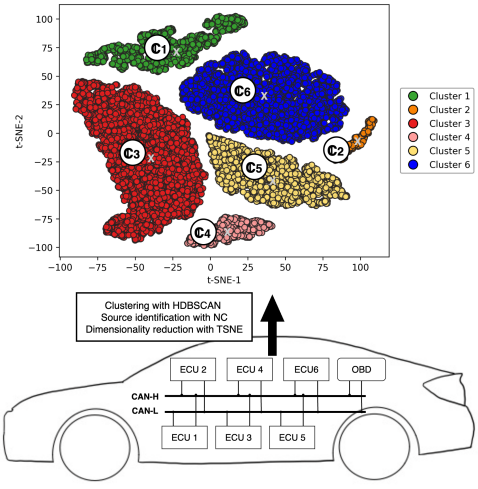
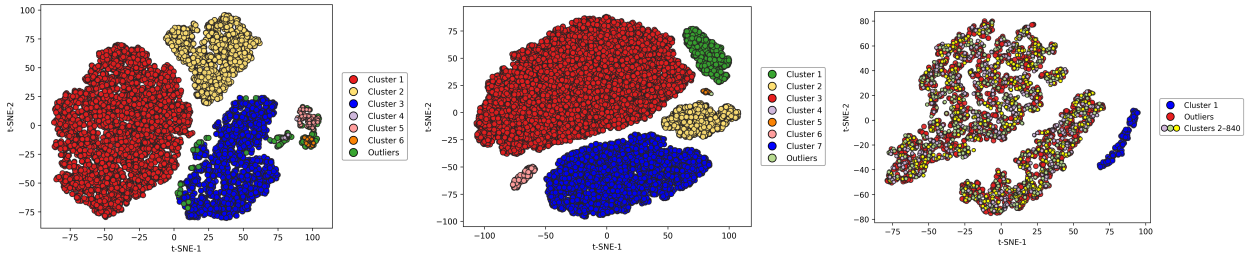


Figure 1: Suggestive depiction of the procedures employed in this work: clustering with HDBSCAN, source identification with Nearest Centroid and dimensionality reduction with TSNE

*Corresponding author: bogdan.groza@upt.ro
patricia.iosif@student.upt.ro (P. Iosif); lucian.popa@aut.upt.ro (L. Popa); bogdan.groza@upt.ro (B. Groza)
ORCID(s): 0000-0003-0829-5467 (P. Iosif); 0000-0001-5357-7776 (L. Popa); 0000-0003-3078-3635 (B. Groza)



(i) Dacia Duster (ii) Ford Ecosport (iii) John Deere

Figure 2: Clustering failures for three vehicles: Dacia Duster, Ford Ecosport and John Deere

introduced in [5] and contains isolated bits collected from 10 vehicles (9 passenger cars and 1 utility vehicle).

In this work, we employ a clustering scheme based on HDBSCAN for determining the ECU responsible for the transmission of each ID and propose a memory-efficient source identification scheme based on Nearest Centroid, as suggested in Figure 1. We use a filtered and aligned version of the ECUPrint dataset [5] that has been recently curated on GitHub[1]. In what follows, we refer to the ECU-ID allocation of this recently published version as the ground truth (GT) for our work. The source code behind the results in this paper is also publicly available[2]. The procedure employed in this work is suggested in Figure 1. We use a filtered and aligned version of this dataset and start by exploring the use of HDBSCAN for clustering such a large volume of voltage data for in-vehicle ECUs (Electronic Control Units) [5]. The main reason for choosing this algorithm is the fact that it does not require the number of clusters to be known in advance like other, more popular algorithms, e.g., K-means. Notably, the number of clusters (ECUs) are known only by the manufacturers, i.e, this is not public information in general. But tuning the parameters of HDBSCAN, despite not requiring the number of clusters, turned out to be a challenge in itself. To serve as a motivating example, we show why the application of this well-known clustering algorithm is not that straight-forward. One naive approach would be employing HDBSCAN in its most basic configuration, i.e., adjusting only its two primary parameters based on a simple assumption: given a car, the minimum cluster size corresponds to the least number of samples associated with any of the vehicle's CAN bus IDs. However, this proves to be insufficient, as highlighted in Figure 2 (i), (ii) and (iii). For the Dacia Duster, a vehicle with 3 known ECUs, this naive approach to HDBSCAN identifies 6 clusters, while for the Ford Ecosport, instead of 4 ECUs, it identifies 7. For the John Deere tractor, the situation is even worse, as it wrongly identifies 841 clusters instead of 3. An additional outliers cluster is present in all three vehicles that we just mentioned. These examples suggest that in order to generalize to a larger pool of vehicles, further tuning is indeed required.

[1] https://github.com/LucianPopaLP/ECUScan

[2] Source code momentarily available for review at the following link, will be moved to a public GitHub following paper acceptance: https://www.aut.upt.ro/~bgroza/projects/hdbscan_nc_on_ecuprint.zip

Clustering is, however, just a first step in designing a system that is actually useful for detecting intrusions inside vehicles. This requires a memory efficient solution that can also cope with environmental changes. In this respect, we use the Nearest Centroid (NC) classifier, an algorithm often overlooked in other works, e.g., [6]. As we later show in the experiments, it exhibits close to 100% true acceptance rate and requires merely 264 Bytes of memory per centroid (including Python-specific metadata). We further use a centroid updating mechanism that makes source identification possible even with variable environmental conditions that impact the variation of the voltage data (that are imminent once the vehicles have been used for a certain period of time). As such, we manage to raise true acceptance rate up to 99.93% for both the Honda Civic and Ford Fiesta on voltage data available after 60 minutes of driving.

The main contributions of this work can be summarized as follows:

- we employ HDBSCAN to provide an improved mapping of IDs to ECUs from the original ECUPrint dataset [5], which matches the ground truth from the aligned version of the ECUPrint dataset, even though it results from an unsupervised clustering algorithm that was not exposed to the labels, acknowledging the four additional ECUs and removing one by assigning the corresponding IDs to another ECU;
- we showcase a parameter-selection approach for HDBSCAN, comparing two well-known approaches for selecting ϵ with an empirical selection;
- we use the Nearest Centroid classifier for source identification, which demonstrates a very high correct acceptance rate and precision, while providing a very small memory footprint, requiring 4–20 times less memory than commonly used algorithms like SVM and RF;
- we use a centroid update mechanism that preserves the performance in cases when vehicles have been subjected to environmental and temperature changes due to driving.

The rest of this paper is structured as follows: Section 2 presents a brief background on the CAN bus as well as related work. Section 3 describes the clustering and

parameter selection methodology, while Section 4 presents the clustering results. Then, in Section 5 we describe our experiments with the Nearest Centroid classifier and study the impact of sampling rate and noise in Section 6. Finally, we state our conclusions in Section 7.

2. Background and Related Works

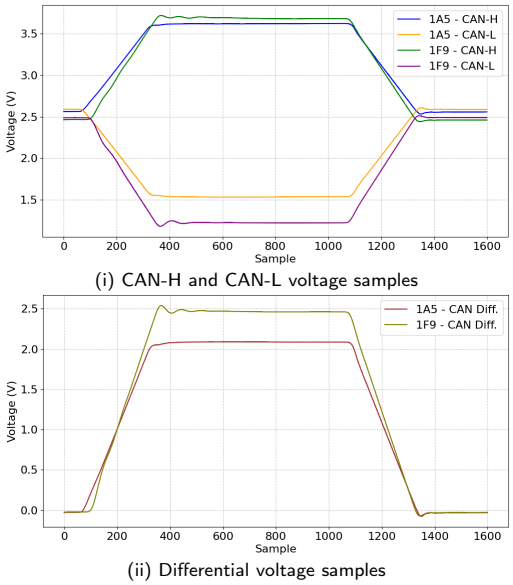
In this section, we provide a quick background for the CAN bus, as well as a summary of the attempts made so far to identify ECUs based on voltage data.

2.1. Background on Controller Area Network

The Controller Area Network was designed by Bosch a few decades ago in order to allow robust communication between electronic and electro-mechanic components inside cars, replacing existing point-to-point configurations. While the initial intention was utilizing them for automotive purposes, they were also adopted by other industries such as aviation and industrial automation. Controller Area Networks are commonly designed as a bus topology that allows multiple nodes to communicate on the same physical interface, which is a pair of twisted wires called CAN-H (CAN-High) and CAN-L (CAN-Low) having two resistors of 120 Ω that interconnect them at the bus ends.

Data is transmitted on the CAN (Controller Area Network) bus using bits that are encoded as states of differential voltage. The bit states correspond to the dominant ('0') and recessive ('1') states of the physical bus. There are two CAN protocols defined that allow maximum bit rates for data transmission. Low-speed CAN is limited to 125 Kbit/s, while high-speed CAN is limited to 1Mbps. CAN bus extensions such as CAN-FD (CAN with Flexible Data rate) and CAN-XL (CAN eXtra Large) have been designed to allow higher bit rates of 5-8 Mbit/s for the first and 20 Mbit/s for the latter. The CAN protocol relevant for this work and further described in this section is the high-speed CAN [7] used for vehicle applications with a nominal bit rate of 500 Kbit/s [8]. The dominant and recessive states of the physical bus for high-speed CAN are represented by a nominal differential voltage of 2.5V and 0.0V for the high-speed CAN protocol that correspond to nominal voltage levels of 3.5V, 1.5V and 2.5V, 2.5V for CAN-H, CAN-L according to ISO11898-2 [7]. Depending on the node, the voltage levels of both CAN-H and CAN-L may vary between 2.0V and 3.0V, while recessive bits are sent under the condition that the differential voltage does not exceed 50 mV. For dominant bits, the voltage level of CAN-H may vary in the range of 2.75V to 4.5V, while the voltage level of CAN-L may be somewhere within 0.5V and 2.25V with the differential voltage in the range of 1.5V to 3.0V.

Bits transmitted on the CAN bus are grouped into different 8 bit fields that are SOF (start-of-frame), arbitration, control, data, CRC (cyclic redundancy check), ACK (acknowledge), EOF (end-of-frame) and IFS (interframe-space). These bit fields are grouped in a specific way to define the CAN message, which is also called the CAN frame. The SOF (start-of-frame) field is a dominant bit used by CAN nodes to start transmission of a CAN frame. The arbitration field contains the frame identifier that is usually called the frame ID. The frame ID can be 11 bits or 29 bits for standard or extended CAN frames. The control field specifies the number of data Bytes that are transmitted in the frame, which is limited to 8 Bytes. The data field contains the actual data for a positive DLC value, followed by the 16-bit CRC (cyclic redundancy check) which has a 15-bit checksum of the previously transmitted bits and a recessive bit as a delimiter. The next one is the ACK (acknowledge) field that contains the acknowledge bit and a recessive bit as delimiter. The acknowledge bit is left recessive by the frame transmitter and sent as dominant by the frame receivers in case the CRC check passed. After frame reception is acknowledged, the end of a CAN frame consists of the EOF (end-of-frame) field of 7 recessive bits which is followed by the IFS (inter-frame-space) field of 3 recessive bits.



(i) CAN-H and CAN-L voltage samples

(ii) Differential voltage samples

Figure 3: Dominant bit samples for CAN-H, CAN-L and differential voltage on two Dacia Duster dominant bits (IDs 1A5 and 1F9)

An example of CAN-H and CAN-L voltage samples for dominant bits and their transition from and to recessive bits is shown in Figure 3 (i). The bits are extracted from two frames, 1A5 and 1A9, transmitted by nodes from the Dacia Duster vehicle, which are included in the dataset evaluated in this work. Although the dominant bit voltage samples for both IDs are higher than 3.5V for the CAN-H line, only those from 1F9 are lower than 1.5V for the CAN-L line. Figure 3 (ii) illustrates the differential voltage that results from the subtraction of CAN-L voltage samples from CAN-H voltage samples. Due to a lower value of the CAN-L samples for the dominant bit, the differential voltage is close to 2.5V for 1F9 while for the other frame it is close to 2.0V.

The bus is in idle state before the CAN communication is started by the nodes. The idle state is represented by the recessive state of the physical bus. Although multiple nodes may start frame transmission at the same time following data

transmitted during the arbitration field, only one node will continue transmission of the remaining bit fields. During arbitration, transmission of a dominant bit will overwrite transmission of a recessive bit due to the wired-AND logic of the CAN bus, i.e., bus state is recessive ('1') only if all nodes transmit a recessive bit. In case multiple nodes start frame transmission at the same time, the node that transmits the frame with the lowest ID value wins the arbitration and continues frame transmission. All other nodes need to wait for the transmission to complete and the bus to be idle again before retrying frame transmission.

Each node has its CAN transceiver connected to the physical communication layer. The transceiver converts the bit state received from the node's CAN controller output (CAN-TX) to a differential voltage. At the same time, it converts the differential voltage from the bus to a bit state that serves as the node's CAN controller input (CAN-RX). This allows each node to verify the physical state of the CAN bus during frame transmission or frame reception. The management of the CAN controller, which is done from the CAN software driver, is part of the application that runs on each CAN node.

2.2. Related Work

Intrusion detection has become popular on vehicular CAN buses [9] and more recently even on CAN buses inside drones [10]. With the increased availability of voltage characteristics datasets, the number of works focusing on such data has also grown in recent years. We use a filtered and aligned version of ECUPrint [5], the largest voltage-based dataset currently available. In the corresponding paper, the authors employ a statistical approach to map CAN bus IDs to ECUs. While most of the existing works focus on sender identification, very few works choose to explore clustering solutions that offer unsupervised mappings between IDs and ECUs based on voltage characteristics. One such work is [6], which uses DBSCAN to map bit slices to sender ECUs on ECUPrint data, discovering two additional ECUs inside the Ford Kuga, a finding further confirmed in both the ground truth and this work, where we use HDBSCAN. However, there are some key differences, namely, one more ECU in the Ford Ecosport, one more ECU in the Ford Fiesta and one ECU less in the Hyundai i20. The authors further propose a sender identification mechanism, concluding that the Random Forest classifier offers the best trade-off between accuracy, memory footprint and speed. Although this method yields an accuracy of 99.68% on the worst-case vehicle, the authors omit experimenting with the Nearest Centroid classifier, which can significantly reduce memory usage, maintaining high accuracy and inference speed. Dini et al. [11] propose a centroid-based classifier for CAN frame source identification, designed to be compact, i.e, it required 128 Bytes per ECU to store each centroid. In their approach, a centroid is defined by two features: the mean values for multiple dominants bits across CAN-H and CAN-L, respectively. They also use the difference of these 2 mean values, however, this feature does not require additional storage. They validate their approach on an experimental setup that emulates a real vehicle (i.e., 8 ECUs), whereas we use a larger dataset spanning 10 cars and 57 ECUs. Moreover, three statistical features (the CAN-H and CAN-L mean values, and their difference) may not be sufficient to separate in a larger pool of vehicles [5]. In another work from the same research group [12], the authors also study other classification methods such as SVM (Support Vector Machine), ANN (Artificial Neural Networks), QDA (Quadratic Discriminant Analysis) and DT (Decision Trees) instead of the centroid-based classifier. They conclude that SVM is the best at satisfying both accuracy and computational constraints. Nevertheless, the two aforementioned solutions are suitable for embedded integration since their functionality was tested on automotive-grade high-end microcontrollers.

Clustering of ECUs based on specific voltage characteristics is studied in [13] through the usage of Bayesian Gaussian Mixture Models (BGMM) coupled with the Bayesian Information Criterion (BIC) to determine the optimal number of clusters. Inference is based on minimizing the squared distances between new data points and cluster centers. GMMs are also used in [14], where the authors design a source ID fingerprinting system, therefore eliminating any need for ID to ECU mapping. Inference is again performed using the distance between new messages and learned centroids. A similar approach is also used in [15], which proposes a clustering and identification system that learns a fingerprint comprising of a mean value and a covariance matrix for each ECU and groups IDs that are minimally distanced by the Mahalanobis distance. Clustering and unsupervised learning has been also used for intrusion detection on regular CAN traffic in [16] and [17].

In terms of works strictly targeting ECU source identification, one of the earliest voltage-based methods is described in [18]. The authors focus on classical signal processing methods. and, although exhibiting some limitations, this work paves the way for further research. In [19], the authors perform extensive experiments with 2 machine learning classifiers on the ECUPrint dataset: Logistic Regression and Random Forests, with improved feature selection. They conclude that the Random Forests classifier offers the best trade-off between accuracy and performance (99% accuracy and below 11 ms inference time). A convolutional neural network (CNN) approach is proposed in [20] and evaluated on the ECUPrint dataset. Concretely, this method experiments with signal processing and machine learning by combining various algorithms and signal transforms. They report the best accuracy of 83% when feeding the differential voltage through the Walsh–Hadamard Transform. Using voltage side-channel measurements from the main power supply of the main controller and multiple CAN transceivers from a Gateway unit, authors of [21] are able to identify genuine transmission of CAN frames with 99.9% accuracy using deep learning methods and with 99.67% accuracy using gradient boosting (XGBoost). Their method was proven to be effective on embedded hardware using AURIX TriCore as the underlying platform with ADC capturing independent

samples at a rate of 4 MHz while the memory usage for their source identification algorithm is of 16.83 kB for the RAM area and 382.kB for the ROM area.

The authors of VIDEN [22] propose a voltage-based intrusion identification algorithm that uses the samples of the acknowledge bit to determine whether a frame transmitter is genuine or not. In the event of two ECUs with similar voltage profiles, the Random Forest classifier is used to identify the correct transmitter with the support of 6 distinct voltage features. Intrusion detection is studied by Choi et al. in [23], with evaluation done on 72 IDs from two vehicles using the machine learning classifiers Linear Support Vector Machine (SVM) and Bagged Decision Trees (BDT). One limitation admitted by the authors is the inability to determine which CAN frame identifier belongs to which ECU – this is precisely the topic that we study here using clustering. A novel detection method that uses only one voltage feature, the bit time, is proposed by [24] with Multinomial Logistics Regression used as the classifier. Another bit-time based IDS is proposed in [25]. This is an online learning solution with two main contributions, an efficient sampling mechanism and a neural network that is periodically re-trained with updated CAN-H voltage data. Single and multiple CAN bit timings together with other voltage features are used in [26] to evaluate source identification accuracy and execution time with several classifiers. Voltage samples from a vehicle with 8 ECUs, collected with a PicoScope at a sampling rate of 20 MHz, are used. Their evaluation shows that Softmax outperforms the other methods with respect to timing while BPNN (Back-Propagation Neural Networks) and RNN (Recurrent Neural Networks) have a slightly better identification performance. A lower sampling rate of 6.25 MHz is proposed in [27] where the authors achieve an identification rate of the sender higher than 99% using the RF (Random Forest) algorithm for classification with less than 10 time and frequency domain features of CAN dominant bits. The same sampling rate of 6.25 MHz is used for CNN classification of CAN transmitters by the authors of [28] with a reported performance of 96.13%.

Other research works that target source identification of CAN frames for intrusion detection study the usage of three machine learning classifiers, Support Vector Machines (SVM), Bagged Decision Trees (BDT) and Neural Networks (NN) [29], achieving the best identification results of 96.48% with the latter. Kneib et al. introduce Logistic Regression as the underlying classifier in their proposal and improve the identification results compared to all previous research work, achieving a true positive identification rate of 99.85% with [30], 99.90% with [31], and 99.98% with [32]. The authors of [33] propose improvements through introduction of reinforcement learning with the outcome of having 20% lower detection timing and 60% less false positives compared to existing benchmarks [22]. Another reinforcement learning improvement comes from [34], where the authors design an authentication system based on both message arrival timing and voltage characteristics. They experiment with reinforcement learning both in isolation and with a deep learning component. For the latter, they report 57%–84% fewer false positives and 34%–85% fewer false negatives for both periodic and non-periodic frames, compared [33]. In [35], a voltage-based IDS is proposed by modeling each ECU with a quasi-linear regression model and using Recursive Least Squares (RLS) for optimization. Slope-monitoring is then used to detect the source ECU of malicious messages. Interestingly, the authors assume that the mapping between CAN bus IDs and ECUs is known beforehand. The authors of [36] propose a more complex intrusion detection method that applies to J1939 communication buses, commonly used in heavy duty and agricultural vehicles. They focus on timing and data analysis to detect adversarial frames in spoofing or masquerade attacks. The identification of illegitimate CAN frame transmitters is proposed in [37], through the use of FeatureBagging-CNN. The mapping between IDs and ECUs is predetermined using clock skews. Another IDS that uses deep learning is proposed in [38]. The authors extract 14 time-domain features from voltage signals to train a deep Support Vector Data Description (SVDD) that works as a binary classifier meant to distinguish between normal and abnormal readings. Yin et al. [39] study overlapped voltage attack identification on an experimental testbed with Long Short-Term Memory (LSTM) autoencoders considering the limitations of state-of-the-art transmitter identification methods [22], [30], [40] to detect this type of attack. They are able to predict voltage waveforms for dominant bits in normal communication scenarios and detect overlapped voltage attacks on the physical CAN bus using these predictions.

Table 1 compares several of the previously mentioned works and our own. Regarding the data on which these works are based, they either use data collected from real-world vehicles or from paper-specific experimental setups. Both approaches have advantages and disadvantages. A laboratory setup is less realistic than data obtained from a vehicle, as it experiences less electrical noise due to missing events such as cranking, braking, relay coupling, etc., however, it is more realistic if the solution is deployed on actual controllers from the setup. There are various features used throughout these works to fingerprint ECUs, including time and frequency domain features. Classifiers vary from centroid-based approaches to more complex machine learning solutions such as SVM and RF. Unsupervised clustering, e.g., DBSCAN or HDBSCAN, is also used in some works to determine the mapping between IDs and ECUs. Regarding the intrusion detection rate, related works report a minimum accuracy between 93.41% to 99.68%. In our paper, we prefer to use the true acceptance rate and false acceptance rate, which are more intuitive, and we obtain the lowest true acceptance rate at 99.93% and the highest false acceptance rate at 0.03% – in the worst case of environmental updates after 60 minutes for two of the cars in the dataset (Ford Fiesta and Honda Civic). The signal acquisition hardware and the sampling rate also vary, including two PicoScope models and two AURIX microcontrollers, with sampling rates ranging from 700 kHz to 500 MHz. Only some of the papers mention

Table 1
Comparison between some Voltage-based ECU Fingerprinting Approaches

Wrk.	Data	Features	Algorithms	Detection rate (min.)	Signal Acquisition	Memory requirements	Update	Noise
[26]	1 car	bit time, plateau time of 5 bit-block, voltage mode	Softmax classifier	96.43%	PicoScope 4444, 20 MHz	unspecified, likely low (11 features)	n/a	✓
[27]	10 cars (ECUPrint)	10 features per vehicle from 3 segments (rising and falling edges, dominant state)	RF	99.61%	PicoScope 5000, 6.25 MHz	unspecified, likely medium (10 features + RF)	n/a	n/a
[6]	10 cars (ECUPrint)	transition corner on rising edge and plateau	DBSCAN & RF	99.68%	PicoScope 5000, 500 MHz	higher memory footprint, Python pickled model 8718–102,766 Bytes	n/a	✓
[11]	lab setup	mean of CAN-H, CAN-L and differential voltage	Centroid-based classifier	99%	AURIX TC375, 700 kHz	very low, 16 Bytes per ECUs (128 Bytes for 8 ECUs)	n/a	n/a
[12]	lab setup	5 features: mean, variance, skewness, kurtosis, hyperskewness	SVM	93.41%	AURIX TC375, 1.3 MHz	low, 1517 Bytes for 8 ECUs	n/a	n/a
Ours	10 cars (ECUPrint)	16 features from differential voltage via PCA	HDBSCAN & NC	99.93%	PicoScope 5000, 500 MHz	low, 128 Bytes per ECU, Python pickled model 2424–4544 Bytes	✓	✓

the memory footprint. Through synthetic estimation, our approach requires around 128 Bytes of memory per ECU with PCA-16. The lowest memory footprint reported in the literature is 16 Bytes per ECU [11]. But this is achieved for a sampling rate of 700 kHz that is equivalent to 1 sample every $1.43\mu s$, while a CAN bus bit lasts around $2\mu s$ on a 500 kbps CAN – amounting to a single sampling point per bit. While this was feasible on the laboratory experimental setup from [11] that had 8 ECUs, the dataset that we used shows that, for the general case of data extracted from 10 vehicles gathering 57 ECUs, this is not enough for reliable separation, as depicted by our clustering results. Therefore, we settled on the aforementioned memory footprint using PCA-16, which we find sufficiently small and achieve a true acceptance rate for intrusion detection over 99.93%. This can be lowered to 16 Bytes per vehicle with PCA-2, and it will work on some of the cars, but there are exceptions, e.g., Ford Kuga and Hyundai ix35. An updating mechanism does not appear to be typically pursued in related works, though it seems necessary for long-term use. In contrast, noise was addressed more consistently.

3. Clustering with HDBSCAN

In this section, we provide an overview of the methodology and how the parameters are selected for the experimental results that follow.

3.1. Tools and Environments

We use a variety of libraries and frameworks for clustering available in Python (version 3.10), including two implementations of Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN). One of these libraries, which has considerable significance for us, is *hdbscan* (`https://hdbscan.readthedocs.io`), which focuses entirely on this particular algorithm. We use it to conduct our experiments on all passenger vehicles from the dataset, i.e., nine out of ten cars. However, when using this library on the heavy-duty John Deere vehicle, no suitable parameters were found to match all its ECUs as detailed in Section 4. Therefore, we continued to explore another implementation option which is available from *Scikit-learn* (`https://scikit-learn.org/stable/`), a library we are already using for dimensionality reduction. This library offers its own variation of the HDBSCAN algorithm. We employ this variation for the task of clustering ECUs inside the John Deere tractor. As mentioned, we also use *Scikit-learn* for dimensionality reduction through the use of Principal Component Analysis (PCA). Moreover, we employ the k-nearest neighbors algorithm made available by this library for parameter tuning in Section 3.4, where we also use the *kneed* (`https://kneed.readthedocs.io/en/stable/index.html`) library to estimate the elbow of certain curves for parameter selection. For 2-dimensional visualization of clustering results, we once again use *Scikit-learn*, through the T-distributed Stochastic Neighbor Embedding (TSNE) method, with the XY components being computed on top of the PCA dimensionality reduction.

3.2. HDBSCAN

The choice of HDBSCAN [41] in our work is firstly justified by the failure of baseline algorithms like K-means. For brevity, we defer the concrete experimental data regarding the failures of K-means on 3 cars to the supplementary material, in Appendix A. Briefly, on the most complex car from our dataset, the Ford Kuga, 6 out of 11 ECUs either have their samples entirely mismatched with other ECUs or only half of these samples are part of the correct clusters. The same observation can be made for 3 ECUs from the Hyundai ix35 and 2 ECUs from the Ford Fiesta. K-means showed the correct clusters for all samples only in 3 of the vehicles from the dataset: Dacia Duster, Opel Corsa and

Honda Civic. It also failed on all other cars that are not mentioned here. Moreover, in the case of K-means, these failures were present despite the fact that we provided the expected number of clusters to the algorithm. Note that HDBSCAN does not accept the expected number of clusters, this is determined internally by the algorithm.

The mere nature of the data, which consists of samples with high density (this is in fact visible from the various plots of the samples), turned us toward a density-based method. Due to imbalances in the dataset, instead of using the classical DBSCAN [42], we decided to adopt HDBSCAN since it handles density in a hierarchical manner. Concretely, DBSCAN is limited to identifying only a fixed density controlled by its ϵ parameter across the entire dataset, and is therefore susceptible to wrongly cluster sparser regions. Since the ECUs from our dataset exhibit significant variations in the number of samples, e.g., ECU 7 from Ford Fiesta contains only 68 samples, while ECU 5 of Ford Ecosport contains 11,842 samples, employing HDBSCAN seemed more natural due to its hierarchical manner of working with varying densities. As for ϵ, the role of the parameter in HDBSCAN is different from that in the original DBSCAN, serving only to merge clusters that are closer than a certain distance, i.e., ϵ. Therefore, we can tune this threshold after the hierarchy has been constructed with the knowledge that the algorithm is not directly biased towards a single density level to better suit each dataset.

Hierarchical Density-Based Spatial Clustering of Applications with Noise [41] is an unsupervised machine learning algorithm that extracts clusters from unlabeled data. In particular, it is able to work with arbitrary-shaped clusters and is very effective at identifying patterns in uneven observation distributions. Given that the method also allows for direct estimation of the number of clusters, it is a suitable approach to determine how many ECUs are in a vehicle, a scenario in which this information is usually known only to the manufacturers. HDBSCAN, as its name suggests, is a hierarchical clustering algorithm that uses point density to identify clusters within a given dataset. This approach relies primarily on the detection of high-density areas within the data and is also efficient in recognizing noise, enabling early outlier detection.

HDBSCAN is canonically defined by a series of parameters:

- `min_samples` – the minimum number of neighbors (self-inclusive) a point needs to be considered a *Core Point*,
- `min_cluster_size` – the minimum amount of samples required to define a single cluster,
- `cluster_selection_epsilon` (ϵ) – a distance threshold that, when used, defines cluster proximity. Clusters that fall below this threshold are merged into one single cluster during the last step of the algorithm.

Some clarifications for the above parameters, based on the mechanism behind HDBSCAN, may be useful. A *Core Point* or a dense point is defined as any sample in the dataset for which at least `min_samples` samples (including the point itself) can be identified inside its designated region, characterized by a radius equal to the *Core Distance*. The *Core Distance* is defined as the distance to the k–th nearest neighbor of a point, where k is equal to `min_samples`. Any point which does not meet this criterion is labeled as an *Outlier* (noise). The distance metric used in this case is the Euclidean distance.

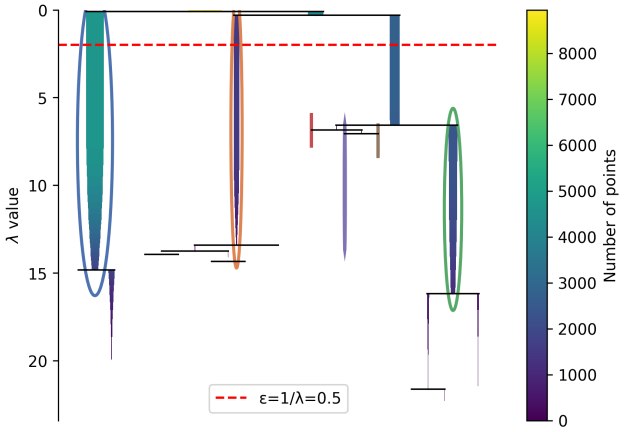


Figure 4: Dendrogram exemplification on Dacia Duster

Consequently, the concept of *Mutual Reachability Distance (MRD)* must be defined. Consider two data points, X_1 and X_2, each with arbitrary dimensionality. The *MRD* between the two points is an indicator akin to a cost function. It penalizes samples that either belong to low-density regions or are significantly distant from one another by maximizing between these two criteria. In other words, the *MRD* between X_1 and X_2 is defined as the maximum between the core distance of X_1, the core distance of X_2 and the actual distance between X_1 and X_2, i.e.,

$$\text{dist}_M(X_1, X_2) = \max\{\text{core}_{\text{dist}}(X_1), \text{core}_{\text{dist}}(X_2), \text{dist}(X_1, X_2)\}.$$

The algorithm works as follows. Initially, it constructs a *Minimum Spanning Tree (MST)*. Assuming all points in the dataset are vertices of a connected weighted graph, the weight given to each edge denotes the *MRD* between the two points. Using Prim's algorihm [43], the corresponding *MST*, which interconnects all nodes at the minimal cost (i.e., the minimum total edge weight), is built. The next phase is transforming the *MST* into a cluster hierarchy. First, the edges are arranged in ascending order before progressively incorporating them into the hierarchy tree, with all data points serving as leaves. This produces a dendrogram, which is a tree-shaped structure, and descending down the tree reveals progressively smaller clusters (or interconnected subtrees). We provide an example of a dendrogram in Figure 4 for the Dacia Duster. The selection of the parameter ϵ, which is used to cut the dendrogram, is discussed in a forthcoming

section. Conversely, climbing the tree reveals clusters that gradually diverge from one another in terms of *MRD*. This is followed by the condensation of the tree into what is known as the *Condensed Tree*. The *Condensed Tree* is derived by navigating the cluster hierarchy and combining clusters that contain fewer points than `min_cluster_size`. The last phase involves choosing final, flat clusters. The stability of each cluster is assessed by examining its longevity, namely the duration for which points persist inside the cluster as the hierarchy is traversed or, alternatively, across various density levels. The stability of each cluster is evaluated against that of its sub-trees, ensuring that only the most stable clusters persist.

The last parameter referenced, ϵ, originates from HDBSCAN's predecessor, DBSCAN [42], however, its meaning differs from the original paper. This parameter is applicable just in the last step of the algorithm, namely during the cluster flattening process. It serves as a distance threshold, leading to the concatenation of clusters when the mutual reachability distance is less than the ϵ threshold.

3.3. HDBSCAN Parameters Overview

As stated, the output of HDBSCAN is controlled by a set of parameters: `min_samples`, `min_cluster_size` and ϵ. For each vehicle, we tune these parameters to better fit each individual data distribution.

- `min_samples` – for each vehicle, we set this value to the minimum number of bits found in an ID that is known to belong to that specific car. We note that the ID distribution in each car is already known at the time when clustering is performed.
- `min_cluster_size` – once again, we set this parameter to the smallest amount of bits that belong to a single ID associated with the current car. Indeed, in the worst case, a cluster can contain a single ID with the minimum number of samples in the dataset.
- `cluster_selection_epsilon` (ϵ) – we use two literature-defined approaches and one empirical selection to determine the best value for this parameter, but essentially, this parameter is determined by the distance between samples.

In Table 2, we show the exact values for `min_samples` and `min_cluster_size`. As an example, for the Dacia Duster, both parameters were set to 9 because the minimum number of samples for an ID is 9. In case of two particular vehicles, i.e. Ford Ecosport and Hyundai i20, while the minimum number of samples for an ID is 1, we set both values to 2, since the *hdbscan* implementation requires the minimum cluster size be larger than or equal to 2. We note that while the allocation of CAN IDs to specific ECUs is unknown, the set of IDs present in a given vehicle is immediately visible in the CAN bus traffic since the IDs are not encrypted. The IDs and the content they carry are also available for many vehicles on several public GitHub pages [44], [45]. The parameters `min_samples` and `min_cluster_size` are the minimum number of bits for an ID since all data frames with the same ID originate from the same ECU. This is because, according to the CAN standard [4], the arbitration process can be won by a single ECU based on the value of the ID. If several ECUs attempt to use the same ID for a data frame, a conflict would occur during arbitration.

Table 2
Values for parameters `min_cluster_size` and `min_samples`

Car	`min_cluster_size` `min_samples`	Justification
Dacia Duster	9	Samples for ID 5DD
Opel Corsa	2	Samples for ID 772
Ford Ecosport	2	Min samples for HDBSCAN
Dacia Logan	6	Samples for ID 69F
Honda Civic	6	Samples for ID 3D7
Hyundai i20	2	Min samples for HDBSCAN
Hyundai ix35	4	Samples for ID 5A2.
Ford Fiesta	2	Samples for ID 2C3
Ford Kuga	2	Samples for IDs 3EA and 420
John Deere	2	Samples for ID 18FEFC21

3.4. HDBSCAN Parameters Determination

Initially, we execute the algorithm without assigning a value to the parameter ϵ (i.e., using the default setting of $\epsilon = 0$). After comparing with the ground truth, we conclude that five vehicles seem to require further tuning: Dacia Duster, Ford Ecopsort, Hyundai i20, Hyundai ix35 and John Deere. We notice that in all cases, the number of clusters is larger than the ground truth, i.e., 7, 8, 10, 8 and 841 as opposed to 3, 5, 6, 6 and 3, respectively. We note that the number of clusters is significantly problematic in the case of the utility vehicle, i.e., the John Deere tractor, due to the very high number of clusters predicted. Since thresholding the mutual reachability distance can reduce the number of clusters by merging certain clusters together, we proceed with searching for appropriate ϵ values.

Firstly, we employ the elbow method [46]. This approach is first introduced in [42] as a heuristic for choosing ϵ and later reiterated in [47]. Since this method originated as an heuristic for DBSCAN, it primarily focuses on the intermediate stages of HDBSCAN, and as such, we refer to it as the *pre-clustering method*. We use the *Scikit-learn* implementation of the k-nearest neighbors algorithm to calculate the k-distance for all points in the dataset. In other words, for every sample, we obtain the distance to the k–th nearest neighbor. For each vehicle, we set the value of k to be equal to `min_samples`. Then, we arrange the distances in ascending order, from samples closest to their k–th nearest neighbor to those furthest away and plot the k-distances. The elbow method indicates that the optimal k-distance corresponds to the point on the curve where a substantial increase becomes noticeable (i.e., the elbow of the curve), which we visually pinpoint. HDBSCAN relies on k-distances in its early phases, when local densities are established through the mutual reachability distance. Given that ϵ is introduced during the final step of the algorithm to flatten the already pruned dendrogram, we can infer that the k-distance is likely to be a poor approximation for ϵ.

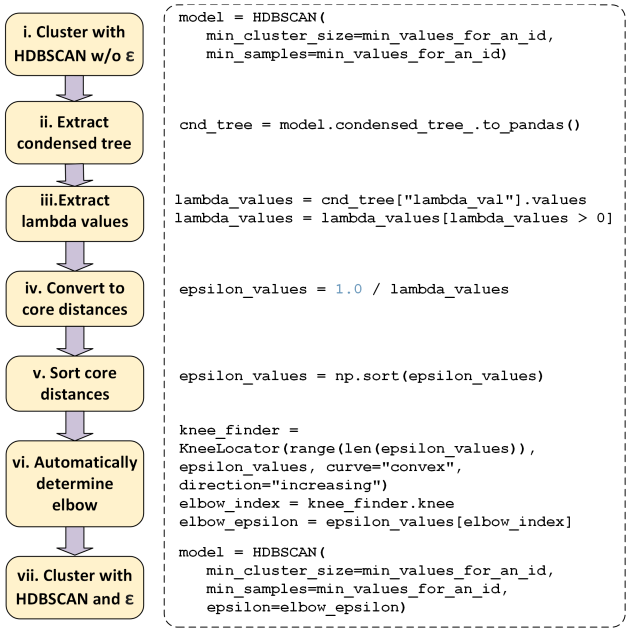


Figure 5: Post-clustering flowchart with Python code

Our experiments show that for Dacia Duster, Ford Ecosport, Hyundai i20 and Hyundai ix35, ϵ is between 0.07 and 0.1. These are quite small thresholds that do not reduce the number of clusters for any of the problematic vehicles. Sub-figures (i) - (iv) from Figure 6 illustrate the elbow curves for each of these cars. For the John Deere, we estimate the ϵ value to be 0.15, which yields 25 clusters as supposed to the original 841 obtained without the use of ϵ.

Secondly, we repurpose the elbow method by using the *Condensed Tree* which HDBSCAN produces before outputting the flattened clusters. This data structure contains all possible clusters in the hierarchy which include more than the `min_cluster_size` samples. Our second way of selecting ϵ, which we refer to as the *post-clustering method*, relies on this particular structure. Concretely, we extract the lambda values corresponding to the *Condensed Tree*, where one lambda value represents the density at which each branch of the tree is formed or, in other words, the density threshold for the branching clusters to exist by itself. Lambda is the inverse of the core distance, i.e., $\lambda = \epsilon^{-1}$. Higher densities, or equivalently smaller core distances, indicate denser clusters.

After filtering out potential erroneous cases, lambda values are converted to core distances before sorting said distances and plotting them in a graph similar to the k-distance plots. This time, we determine the elbow automatically, using the `KneeLocator` class from the *kneed* library. This implementation selects the elbow by locating the curve's point of maximum curvature. This process can be visualized in Figure 5, where we describe the following 7 steps: (i) cluster with HDBSCAN without an explicit value for ϵ, (ii) extract condensed tree structure, (iii) extract lambda values, (iv) convert them to core distances, (v) sort the core distances, (vi) automatically determine the elbow using the *kneed* library, (vii) cluster again with HDBSCAN providing the computed value for ϵ.

Sub-figures (i)–(iv) from Figure 7 show the elbow curves for our four relevant vehicles: Dacia Duster, Ford Ecosport, Hyundai i20 and Hyundai ix35. Resulting ϵ values are 0.22, 0.1631, 0.1518 and 0.3071, respectively. We note that the points of maximum curvature for these particular plots correspond to the points on the curve where values begin dispersing, i.e., where a distribution shift appears. This approach improves the results for one car, namely, Dacia Duster. For this vehicle, the number of clusters is reduced from 7 to 3, which matches with our pre-defined ground truth. We note that for the John Deere tractor, this method is not applicable, due to the *Scikit-learn* implementation of HDBSCAN not providing access to the condensed tree structure.

Thirdly, after the post-clustering method, which is algorithmic (based on Python functions from the *kneed* library), for the empirical selection, we inspect the elbow plotted for each car and manually toggle the value of ϵ starting from the location where the points become sparse. While finding a unique ϵ for all cars was not possible, we ended up with three values that cover all cars (i.e, 0.15, 0.35 and 0.5). This empirical selection reflects the observation that vehicles with closer clusters require smaller values for ϵ, while vehicles with more dispersed clusters require bigger values. In other words, as we increase the distance threshold ϵ, the number of clusters decreases. Therefore, cars where clusters are closer together require a smaller ϵ for separation, while cars with widely dispersed clusters need a larger value.

The selected values for ϵ, are summarized in Table 3, based on the pre-clustering and post-clustering methods, as well as the empirical determination. We note that, even for this empirical experiment, the Hyundai ix35 still has one more cluster beyond the six identified in the ground truth. This cluster acts as an outlier-like cluster, with just five samples. The same applies to the John Deere tractor, where the algorithm still outputs 25 clusters where 3 main clusters stand out while 22 other clusters act as outliers. We acknowledge that the empirical selection is in fact a trial-and-error approach that involves a human in the loop, and propose the post-clustering approach as a fail-safe mechanism when human feedback is not obtainable.

4. Clustering Results

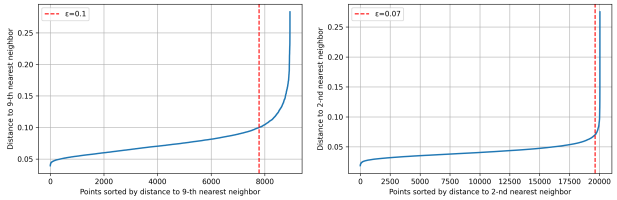
In this section we first introduce the performance metrics for cluster separation and then present the results that we achieve with HDBSCAN.

4.1. Metrics for Classification and Cluster Proximity

We use the following performance metrics with respect to the ground truth in order to assess the effectiveness of HDBSCAN. Let the class variable $c \in \{\text{ECU}_1, \text{ECU}_2, ..., \text{ECU}_k\}$ for k clusters (ECUs) in the car for which we run the experiments. For every class c, we define each classification event type for a sample as follows:

Table 3
Selected values for epsilon and the number of clusters according to several methods

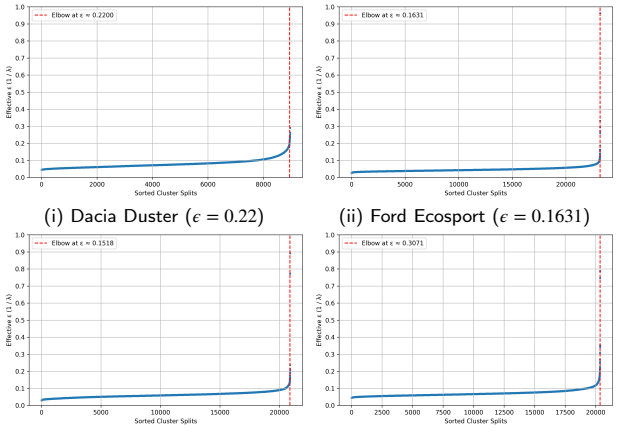
Car	ϵ unspecified	Pre-clustering method	Post-clustering method	Empirical determination	GT
Dacia Duster	Clusters = 7	Clusters = 7, $\epsilon = 0.1$	Clusters = 3, $\epsilon = 0.22$	Clusters = 3, $\epsilon = 0.5$	3
Opel Corsa	Clusters = 4	Clusters = 4, $\epsilon = 0.07$	Clusters = 4, $\epsilon = 0.1218$	Clusters = 4, $\epsilon = 0.5$	4
Ford Ecosport	Clusters = 8	Clusters = 8, $\epsilon = 0.07$	Clusters = 8, $\epsilon = 0.1631$	Clusters = 5, $\epsilon = 0.5$	5
Dacia Logan	Clusters = 6	Clusters = 6, $\epsilon = 0.075$	Clusters = 6, $\epsilon = 0.1369$	Clusters = 6, $\epsilon = 0.15$	6
Honda Civic	Clusters = 6	Clusters = 6, $\epsilon = 0.09$	Clusters = 6, $\epsilon = 0.1360$	Clusters = 6, $\epsilon = 0.15$	6
Hyundai i20	Clusters = 10	Clusters = 10, $\epsilon = 0.075$	Clusters = 9, $\epsilon = 0.1518$	Clusters = 6, $\epsilon = 0.5$	6
Hyundai ix35	Clusters = 8	Clusters = 8, $\epsilon = 0.1$	Clusters = 8, $\epsilon = 0.3071$	Clusters = 7, $\epsilon = 0.35$	6
Ford Fiesta	Clusters = 7	Clusters = 7, $\epsilon = 0.06$	Clusters = 7, $\epsilon = 0.1201$	Clusters = 7, $\epsilon = 0.15$	7
Ford Kuga	Clusters = 11	Clusters = 11, $\epsilon = 0.06$	Clusters = 11, $\epsilon = 0.1207$	Clusters = 11, $\epsilon = 0.15$	11
John Deere	Clusters = 841	Clusters = 24, $\epsilon = 0.15$	N/A	Clusters = 24, $\epsilon = 0.15$	3



(i) Dacia Duster ($\epsilon = 0.1, k = 9$) (ii) Ford Ecosport ($\epsilon = 0.07, k = 2$)

(iii) Hyundai i20 ($\epsilon = 0.075, k = 2$) (iv) Hyundai ix35 ($\epsilon = 0.1, k = 4$)

Figure 6: Elbow determination with pre-clustering method



(i) Dacia Duster ($\epsilon = 0.22$) (ii) Ford Ecosport ($\epsilon = 0.1631$)

(iii) Hyundai i20 ($\epsilon = 0.1518$) (iv) Hyundai ix35 ($\epsilon = 0.3071$)

Figure 7: Elbow determination with post-clustering method

- **TP$_c$** - the number of samples that actually belong to class c classified as such,
- **TN$_c$** - the number of samples that do not belong to class c actually classified as such,
- **FP$_c$** - the number of samples that do not belong to class c classified as belonging to class c,
- **FN$_c$** - the number of samples that actually belong to class c but are classified as belonging to another class.

While these metrics are typically used for supervised classification, they can also be applied to clustering when a ground truth is available. In fact, the well-known and widely-used Fowlkes–Mallows index [48], used to asses the similarity between two clusterings, is based exactly on TP, FP, and FN. Since our ultimate goal is the design of an IDS based on the clustering from HDBSCAN, we continue with metrics that have a clearer meaning in our context. From the data gathered from all the experiments, for each class c we define the following per class metrics:

- **True Acceptance Rate (TAR)** or Recall – is the number of correctly identified samples that belong to class c divided by the total number of samples identified as belonging to class c, including samples that are wrongly classified as belonging to class c, i.e.,

$$\mathrm{TAR}_c = \mathrm{Recall}_c = \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c}.$$

- **False Acceptance Rate (FAR)** or False Positive Rate – is the number of incorrectly identified samples that belong to class c divided by the total number of samples identified as belonging to class c, i.e.,

$$\mathrm{FAR}_c = \mathrm{FPR}_c = \frac{\mathrm{FP}_c}{\mathrm{FP}_c + \mathrm{TN}_c}.$$

- **False Rejection Rate (FRR)** or Miss Rate – is the number of samples belonging to class c that are misplaced in another class divided by the total number of samples belonging to class c, i.e.,

$$\mathrm{FRR}_c = 1 - \mathrm{TAR}_c = 1 - \mathrm{Recall}_c = \frac{\mathrm{FN}_c}{\mathrm{TP}_c + \mathrm{FN}_c}.$$

- **Precision (PRE)** – is the number of samples belonging to class c divided by the total number of samples labeled as belonging to class c, including samples that belong to other classes in the ground truth, i.e.,

$$\mathrm{PRE}_c = \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c}.$$

HDBSCAN may group samples from distinct ECUs under the same label and in this case we consider that the ECU identified by HDBSCAN is the ECU which has the majority of samples. A situation in which multiple ECUs have the same number of samples (majority) under the same label did not occur, but if this happens, we can take the first ECU as the correct class for that cluster.

For a crisper view on distance between the identified clusters, having the centroid of cluster c denoted as

$$\mathbb{C}_c = \frac{1}{|c|} \sum_{x_i \in c} x_i,$$

we also introduce the following proximity metrics:

- **Intra-cluster Mean Distance** or μ_c – is the mean distance between the centroid of a cluster and the individual points inside that cluster, with the centroid being defined as the central point of the cluster, i.e.,

$$\mu_c = \frac{1}{n} \sum_{i=1}^{n} \text{dist}(p_i, \mathbb{C}_c),$$

  where n is the total number of points inside the cluster and $\mathbb{C}_c$ is the centroid of the current cluster.

- **Intra-cluster Standard Deviation** or σ_c – is the variation from the mean of the distances between the centroid of a cluster and the individual points inside that cluster, i.e.,

$$\sigma_c = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\text{dist}(p_i, \mathbb{C}_c) - \mu_c)^2},$$

  where n is the total number of points inside the cluster and $\mathbb{C}_c$ is the centroid of the current cluster.

- **Distance to Nearest Centroid** outside of cluster c or d^c_{min} – is the minimum distance between the centroid of cluster c and the centroid of any other cluster, i.e.,

$$d^c_{\text{min}} = \min_{i \neq c} \|\mathbb{C}_i - \mathbb{C}_c\|, \forall i \in \{1, \ldots, k\}, i \neq c,$$

  where k is the number of clusters, $\mathbb{C}_c$ is the centroid of the current cluster, $\mathbb{C}_i$ is the centroid of cluster i.

We note that these metrics are computed after applying the dimensionality reduction with PCA. Moreover, we define the following proximity heuristic, applicable to all vehicles in our dataset: for a well separated cluster, the nearest centroid should be further away than two standard deviations from the intra-cluster mean distance. This can be formalized as follows:

$$d^c_{\text{min}} > \mu_c + 2\sigma_c.$$

4.2. Experimental Results

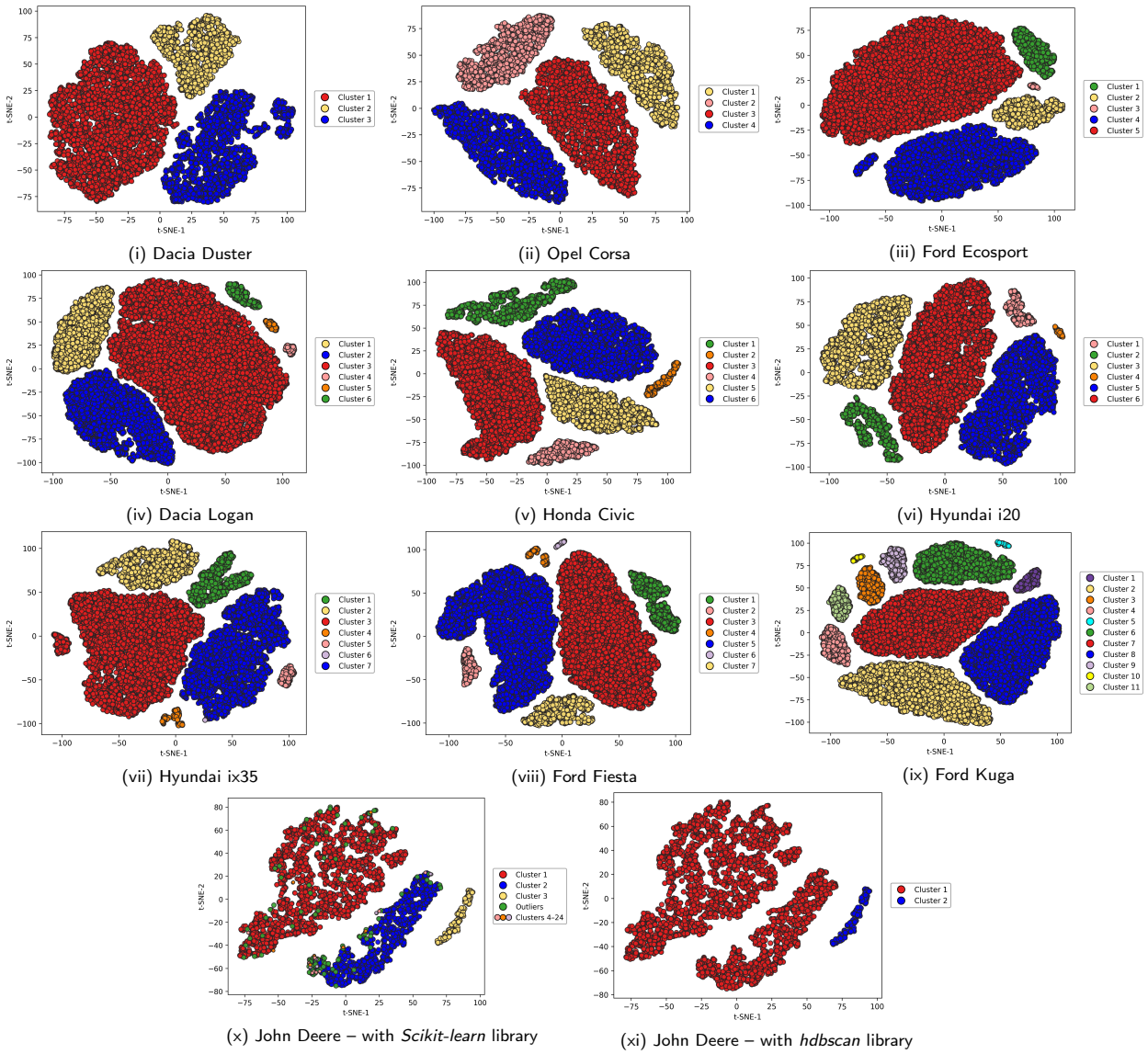
We conduct our final experiments with the following HDBSCAN configuration: for each vehicle, we set the values of `min_samples` and `min_cluster_size` as described in Table 2. For ϵ, the empirically determined values shown in Table 3 seem to be the most suitable. For brevity, the tables containing the complete results for each car are moved to the supplementary material, in Appendix B, and in what follows we discuss each car individually.

We first analyze results for the Dacia Duster using our defined set of metrics. A visual representation of the 3 resulted clusters is shown in Figure 8 (i). For both TAR and PRE, we obtain values of 100%. This equates to FRR and FAR being 0%. In terms of proximity, all 3 clusters respect our heuristic, with the nearest cluster being further away than 2 standard deviations from the mean. We also identify the following particularity for this vehicle, with respect to ECU 2: on the bit plateau, ID 284 displays a fluctuation of around 50 mV. Conversely, the other IDs from ECU 2, i.e., 244, 285, and 354, show a fluctuation of only 20 mV. This can be visualized in Figure 9 (i). We conclude that this phenomenon led other configurations of the algorithm, i.e., without ϵ and with ϵ determined by the pre-clustering and post-clustering methods, to identify additional clusters to appear in this region. This is illustrated in Table 3.

Regarding Opel Corsa, Dacia Logan and Honda Civic, their corresponding cluster-level visualizations can be found in Figure 8, sub-figures (ii), (iv) and (v). Results are similar to those obtained for the Dacia Duster, i.e., both TAR and PRE report values of 100%, while FRR and FAR remain 0%. In other words, for these vehicles, there is no difference between the ground truth and the clustering we performed. The proximity heuristic is met for all three vehicles. Moreover, we report the same observations for the Hyundai i20, Ford Fiesta and Ford Kuga, with 2-dimensional visualizations being displayed in Figure 8, sub-figures (vi), (viii) and (ix), respectively.

Next, we expand on our clustering results for the Ford Ecosport, with a visual representation presented in Figure 8 (iii). The results are similar to the vehicles we have analyzed before: both TAR and PRE are 100% with FRR and FAR remaining 0%. The proximity heuristic is also met once again. However, we note one particularity for this car: visually, ECU 4 appears to be split into 2 individual groups, one much smaller than the other. We plot the bits belonging to this particular ECU in Figure 9 (ii) and observe that 251 out of 5794 bits have a larger bit time, i.e., they take approximately 10 ns longer on average. This behavior could be easily explained by the CAN standard. Namely, when a node sends a message over the CAN bus, some messages might require more time for transmission. Consequently, the transmitting node continually verifies the state of the bus and, if necessary, recalculates its bit time to reflect this state.

For the Hyundai ix35, the clustering visualization is shown in Figure 8 (vii). With the exception of one ECU, our metrics remain consistent with those of the previous vehicles. For ECU 4 TAR is 99.92%, wheras FRR is 0.08%.



(i) Dacia Duster (ii) Opel Corsa (iii) Ford Ecosport

(iv) Dacia Logan (v) Honda Civic (vi) Hyundai i20

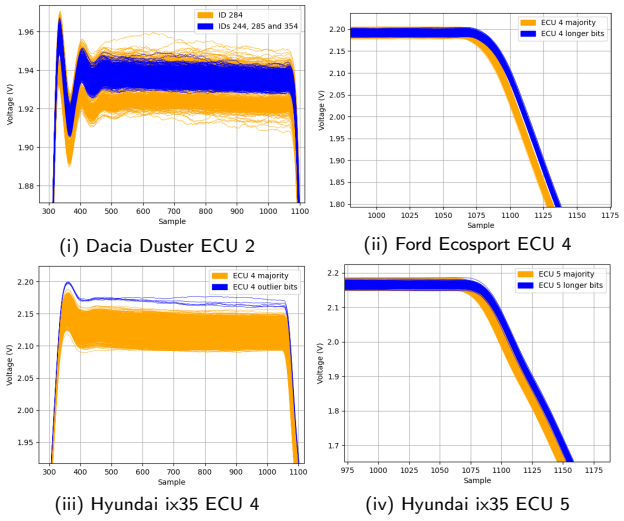
(vii) Hyundai ix35 (viii) Ford Fiesta (ix) Ford Kuga

(x) John Deere – with *Scikit-learn* library (xi) John Deere – with *hdbscan* library

Figure 8: Clustering visualization with TSNE for each vehicle in the ECUPrint dataset

This is due to the Hyundai ix35 clustering having an extra ECU that contains just 5 samples. For this ECU, we report a FAR value of 100%, since these bits denote 5 outliers. The plateau of all bits belonging to ECU 4, i.e., based on ground truth, is shown in Figure 9 (iii), revealing an approximately 40 mV difference between the highest bits in each grouping. Clearly, from a percentage perspective, these outliers contribute very little. They can be explained, for example, by voltage differences that may manifest on the CAN bus while the vehicle is being turned on. In terms of proximity, ECU 4 does not meet our heuristic, however this can be explained by its proximity to the 5-outlier cluster. Additionally, a behavior like the one seen on Ford Ecosport data is also identified here. Specifically, 291 out of 9334 bits from ECU 5 have a larger bit time. While all samples from ECU 5 are grouped by the algorithm, there is a clear separation between 2 different bit times as seen in Figure 9 (iv). On average, some samples are around 13ns longer.

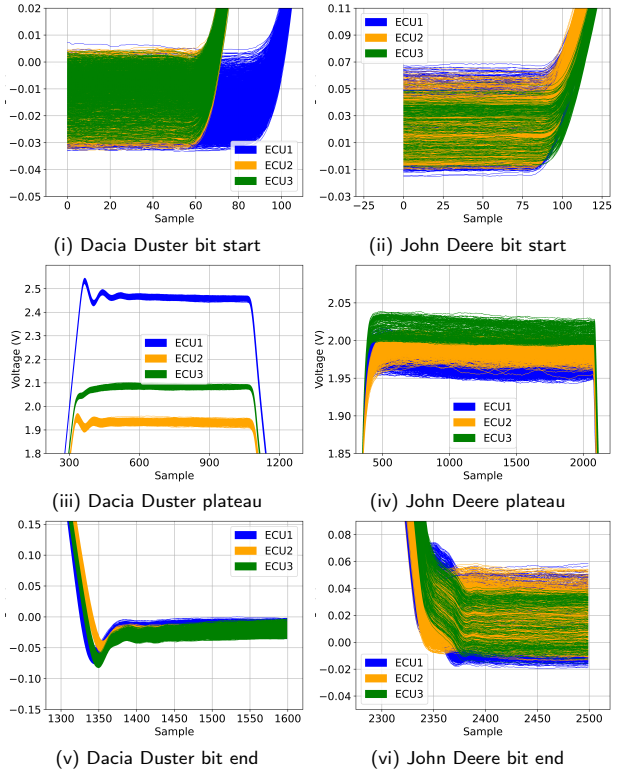
For the John Deere tractor, the visual clustering representation is shown in Figure 8 (x). Only one ECU can be matched one-to-one with the ground truth, i.e., ECU 3. For ECU 1 and ECU 2, we have TAR values of 99.92% and 91.55%, respectively while the FRR is 0.08% and 8.45% respectively. For this vehicle, the algorithm reports 21 very small clusters, i.e, maximum 4 samples, spanning a total of 52 samples we consider outliers. Therefore, all these clusters are negligible and we report a FAR value of 100%. The algorithm also outputs a 99-sample actual Outliers cluster, which we treat in the same exact manner. We note that this particular evaluation was conducted using the *Scikit-learn* implementation of HDBSCAN, instead of the *hdbscan* implementation used beforehand. Moreover, dimensions were reduced to 8 principal components instead of 16. When using the same configuration used for the other vehicles, the algorithm outputs 2 clusters instead of 3, effectively merging together ECU 2 and ECU 3. This can be seen in Figure 8



(i) Dacia Duster ECU 2 (ii) Ford Ecosport ECU 4

(iii) Hyundai ix35 ECU 4 (iv) Hyundai ix35 ECU 5

Figure 9: Edge cases and vehicle particularities



(i) Dacia Duster bit start (ii) John Deere bit start

(iii) Dacia Duster plateau (iv) John Deere plateau

(v) Dacia Duster bit end (vi) John Deere bit end

Figure 10: Separability comparison between Dacia Duster and John Deere

(xi). As far as the proximity heuristic is concerned, none of our 3 main clusters meets the heuristic when computing the metrics on the reported clusters. The closest clusters to the main ones are actually the outlier clusters, which are in fact embedded in the main ones in the 2D visualization from Figure 8 (x). By computing the proximity metrics between the 3 main clusters (these are included in Appendix B from the supplementary material), ECU 2 and ECU 3 will meet the proximity heuristic, while ECU 1 still does not, which can be explained by the increased similarity of the bits. To serve as a visual argument, in Figure 10, we compare the separability of samples from the Dacia Duster with samples from the John Deere tractor on different regions of the signal. On both vehicles, at the beginning of the rising edge, we notice low separability as shown in Figure 10 (i) and (ii). The plateau areas displayed in Figure 10 (iii) and (iv), illustrate a clear separation on the Dacia Duster, while the separation is blurred on the John Deere. Still, ECU 3 from John Deere does appear to have a slightly higher voltage level compared to the other two. In contrast, at the end of the falling edge, ECUs from both vehicles exhibit specific signal patterns that are easy to distinguish as depicted in Figure 10 (v) and (vi). Coupled with the fact that the proximity heuristic is not met for ECU 1, we conclude that ECU 3 is better separated than ECU 1 and ECU 2.

Regarding the cluster proximity metrics, the heuristic offers an overview of the distance to the nearest centroid d^c_{min} in relation to the cluster intra-distance. Briefly, we note that in cases where the heuristic fails, e.g., the John Deere tractor, the intra-cluster mean distance μ_c is greater than 0.3, signaling that the samples inside these particular clusters are much more spread out than in cases when the heuristic is met. This is further confirmed if we look at the intra-cluster standard deviation σ_c which is greater than 0.15 when the condition is not met. The nearest centroids also appear to be as close as 0.372. To serve as a comparison, ECU 1 in the Honda Civic also has σ_c above 0.15 but the nearest centroid is further out at 4.260. This suggests that, typically, the heuristic fails in cases where we have close, sparse clusters, a case best illustrated on the John Deere tractor. The tables from the supplementary material (Appendix B) contain the distance metrics as well for all cars.

To offer a brief quantitative summary of the clustering results, we align them with the ground truth from the dataset in Table 4. Concretely, our mapping corroborates the ground truth, with the clustering scheme identifying outliers for only two vehicles: Hyundai ix35 (5 outliers out of 19,856 samples) and John Deere (151 outliers out of 4,021 samples). All other vehicles exhibit a perfect matching with the ground truth. We also add a qualitative scale for the ease of cluster separation for each vehicle based on the cluster proximity metrics: intra-cluster mean distance μ_c and the distance to the nearest centroid outside the cluster d^c_{min}. As such, we classify the cars as follows: (i) cars with low separability have their intra-cluster mean distance below 2 times the distance to the nearest centroid outside the cluster: $d^c_{\text{min}} < 2\mu_c$, (ii) cars with medium separability are characterized by a distance to the nearest centroid outside the cluster that is between 2 and 10 times the intra-cluster mean distance: $2\mu_c < d^c_{\text{min}} < 10\mu_c$ and (iii) cars with high separability have the distance to the nearest centroid outside the cluster larger than 10 times the intra-cluster mean distance: $d^c_{\text{min}} > 10\mu_c$. Only two vehicles in the dataset have low separability: the Hyundai ix35 and the John Deere tractor, i.e., the cars for which the clustering scheme ran into outliers. All other vehicles, with no outliers, have either high or medium separability.

Table 4
Qualitative summary of the clustering results from this work

			HDBSCAN results	
Car	**ECUs**	**Samples**	**GT match**	**Observation**
Dacia Duster	3	8,942	✓	high separability
Opel Corsa	4	9,131	✓	high separability
Ford Ecosport	5	20,044	✓	med. separability
Dacia Logan	6	31,297	✓	med. separability
Honda Civic	6	14,567	✓	med. separability
Hyundai i20	6	17,767	✓	med. separability
Hyundai ix35	6	19,856	✓	low separability (5 outliers)
Ford Fiesta	7	21,729	✓	med. separability
Ford Kuga	11	28,024	✓	med. separability
John Deere	3	4,021	✓	low separability (151 outliers)

Regarding the low separability in the Hyundai ix35 and John Deere tractor, several contributing factors can be envisioned, for example: hardware characteristics, the impedance of the communication line, external interferences, etc. We reviewed the service manual of a similar John Deere tractor [49], which contains three ECUs (the specific John Deere model is not mentioned in the ECUPrint dataset) and according to the manual, the ECUs are connected using a standard CAN bus, terminated with 120 Ω resistors, and employ a twisted quad cable, where the primary supply voltage and ground wires serve as a magnetic shield for the CAN differential pair. This setup indicates that interferences on the tractor's CAN bus are adequately managed and minimal external noise is expected – a conclusion supported by the bit plots of the collected data as well. A comparable conclusion likely holds for the Hyundai ix35, where the closest service manual that we could retrieve is from a Hyundai Tucson [50], a closely related vehicle to the ix35, and shows the same standard CAN bus termination with 120 Ω resistors. Since no significant differences were observed in the wiring, both vehicles employ standard CAN termination resistors, and little to no noise was detected at the bit level, the observed similarity is most likely attributable to comparable ECU hardware characteristics. The available documentation does not specify the CAN controllers used in the cars and physical inspection is currently out of reach. Fortunately, even with these bit-level similarities, the clustering algorithm is still able to distinguish between the ECUs with minor performance degradation (a small number of outliers).

5. Efficient Identification with Nearest Centroid

In this section, we explore the identification of in-vehicle ECUs with the Nearest Centroid classifier, which has the advantage of an extremely small memory footprint. Then, we discuss the centroid update procedure which is needed in order to cope with environmental variation. Since the ECUPrint dataset contains environmental changes only for two vehicles, i.e., Honda Civic and Ford Fiesta, the results in this section are focused on these cars alone.

5.1. Nearest Centroid Identification

Real-world application requires efficient solutions. HDBSCAN, with its effective density-based mechanism, has helped us determine the right number of clusters, and now, the Nearest Centroid (NC) classifier [51] is perfect for deploying an efficient identification mechanism. The Nearest Centroid classifier is often overlooked, but it is one of the best when it comes to memory footprint, while its computational demands scale linearly with the number of classes. This is because the classifier requires just one centroid for each class. Essentially, at inference time, the algorithm compares new samples with known centroids and assigns each sample to the closest one using the Euclidean distance. For example, in the case where PCA-16 is used, there are 16 components for each centroid and 11 centroids for the most complex car in our dataset. While determining the correct number of clusters with HDBSCAN required extensive parametrization studies as shown in Section 3, with each vehicle having its own distinct set of parameters, the Nearest Centroid algorithm does not need parametrization, it only requires the mapping between IDs and ECUs to extract the centroid corresponding to each ECU.

To better showcase the advantages of using the Nearest Centroid classifier, in Table 5, we present classification results with NC and two other classification algorithms: Support Vector Machine (SVM) and Random Forests (RF). We analyze two key vehicles in our dataset: Honda Civic and Ford Fiesta. When using NC, correct identification, i.e., TAR, is 100% for 5 out of 6 ECUs and 6 out of 7 ECUs, respectively. Likewise, in both cases, PRE is 100% for all but one ECU. In particular, for Honda Civic, ECU 2 is the only unit with TAR below the maximum: 99.99%, and a corresponding FRR of 0.01%. In other words, 0.01% samples are misclassified as belonging to ECU 4. Consequently, for ECU 4, we report 99.99% PRE and 0.01% FAR. For Ford Fiesta, we achieve 99.98% TAR and 0.02% FRR for ECU 2, i.e., 0.02% samples are misplaced as originating from ECU 3. For ECU 3, we reach 99.20% PRE and 0.01% FAR. This equates to minimal misidentification when using the NC classifier, which is crucial for designing a secure source identification mechanism or an IDS, where close to 100% correct identification rate is required. Regarding the other two classifiers, SVM behaves exactly the same as NC, while RF has a negligible decrease of 0.01%, for some of the metrics on the Honda Civic. The same fluctuations of around 0.01% can be seen on the Ford Fiesta and the highest variation is an increase by 0.61% in PRE for ECU 3, slightly favoring SVM this time. However, these small fluctuations in performance are almost negligible and can be attributed to random sampling, a reason for which we consider that detection performance cannot be the ultimate criterion for choosing between these classifiers. We note that these evaluations were performed using a training subset of only 20 randomly sampled bits per ECU, in order to identify a size margin for the training set that is both small enough so data can be collected quickly in a real-world application and

Table 5
Results and memory footprint with NC, SVM and RF for Honda Civic and Ford Fiesta (20 training samples)

Car	ECU	NC				SVM				RF			
		TAR_C	FRR_C	FAR_C	PRE_C	TAR_C	FRR_C	FAR_C	PRE_C	TAR_C	FRR_C	FAR_C	PRE_C
Honda Civic	ECU 1	100%	0%	0%	100%	100%	0%	0%	100%	99.99%	0.01%	0%	100%
	ECU 2	99.99%	0.01%	0%	100%	99.99%	0.01%	0%	100%	99.99%	0.01%	0%	100%
	ECU 3	100%	0%	0%	100%	100%	0%	0%	100%	100%	0%	0.01%	99.99%
	ECU 4	100%	0%	0.01%	99.99%	100%	0%	0.01%	99.99%	100%	0%	0.01%	99.99%
	ECU 5	100%	0%	0%	100%	100%	0%	0%	100%	99.99%	0.01%	0%	100%
	ECU 6	100%	0%	0%	100%	100%	0%	0%	100%	100%	0%	0%	100%
Memory footprint		**3216 Bytes**				**15,154 ± 298 Bytes (4.71× increase)**				**65,280 Bytes (20.3× increase)**			
Ford Fiesta	ECU 1	100%	0%	0%	100%	100%	0%	0%	100%	100%	0%	0.01%	99.99%
	ECU 2	99.98%	0.02%	0%	100%	99.99%	0.01%	0%	100%	99.99%	0.01%	0.01%	99.99%
	ECU 3	100%	0%	0.01%	99.20%	100%	0%	0.01%	99.81%	99.99%	0.01%	0.01%	99.97%
	ECU 4	100%	0%	0%	100%	100%	0%	0%	100%	100%	0%	0%	100%
	ECU 5	100%	0%	0%	100%	100%	0%	0%	100%	100%	0%	0%	100%
	ECU 6	100%	0%	0%	100%	100%	0%	0%	100%	100%	0%	0.01%	99.99%
	ECU 7	100%	0%	0%	100%	100%	0%	0%	100%	100%	0%	0%	100%
Memory footprint		**3480 Bytes**				**19,815 ± 411 Bytes (5.69× increase)**				**66,080 Bytes (18.99× increase)**			

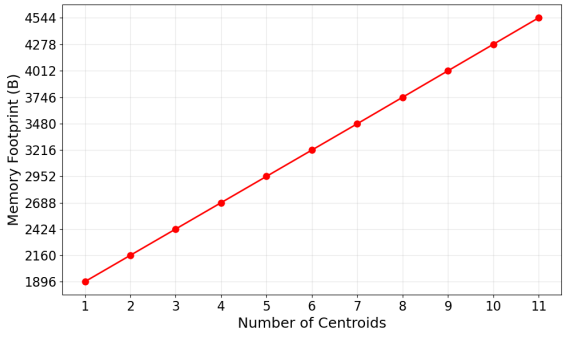


Figure 11: Memory footprint growth trend

representative enough to yield a correct identification rate as close to 100% as possible.

As far as the memory footprint of the NC model is concerned, we also show this metric in Table 5, measured in Bytes. For NC, on a single vehicle, only the coordinates that define the centroids must be saved. Since we reduce our feature-space using PCA-16, this means 16 coordinates are saved for each centroid. As shown in Figure 11, the model's memory footprint grows linearly with the number of centroids. As such, Honda Civic requires 3216 Bytes, while Ford Fiesta requires 3480 Bytes. Meanwhile, the other two classifiers require substantially more memory, with SVM having an approximate memory footprint of 15,154 Bytes on Honda Civic and 19,815 Bytes on Ford Fiesta. This is an approximate measure due to the fact that SVM is based on support vectors, which may change on each run along with the training data. RF requires an even larger footprint, with 65,280 Bytes needed for the Honda Civic and 66,080 Bytes for the Ford Fiesta. With the goal of employing a lightweight, highly accurate and memory efficient solution in mind, we conclude that the Nearest Centroid classifier is the best choice since it offers a significantly lower memory footprint while the performance does not degrade (the experiments showed at most a 0.01% decrease in accuracy which is likely attributable to random sampling effects). We note that these memory estimates are only for the actual RAM size, approximated recursively (i.e., taking into account all stored objects) through the use of the *joblib*[3] and *pympler*[4] Python libraries. When running with different distributions of Python, we notice various memory footprints due to distinct encoding of the centroids. In some cases, the memory actually halved, but even the previous memory footprints are low enough. Nevertheless, we synthetically estimate that the actual memory required for a centroid is 128 Bytes, i.e., 8 Bytes required for 16 coordinates (obtained via PCA-16). Python libraries such as *joblib* and *pympler* also add specific metadata, e.g., variable names, along with the actual centroid coordinates.

The results we report for NC are averaged per ECU across 1000 runs, with training and test splits randomly sampled at each run, as we wished to gain as much confidence as possible in the reported metrics. For SVM and RF, which are clearly disadvantaged by their high memory footprint and used just for comparison, we average the results over 100 runs, to avoid unnecessary long experiments on our computer. To run on an embedded device, one would need the full software stack. Since our current implementation runs on offline data anyway, we leave that as a subject for future work.

5.2. Addressing Environmental Variations with Centroid Updates

Although they excel in the accuracy of recognizing an ECU, voltage techniques are known to be highly sensitive to environmental changes. Fortunately, these changes are gradual for the most part and local fluctuations are an exception, but a fingerprint update mechanism is still necessary. In this respect, the Nearest Centroid algorithm is an excellent choice since it offers an immediate mechanism to update the fingerprints, i.e., the centroid of each underlying class.

[3] https://joblib.readthedocs.io/en/stable/
[4] https://pympler.readthedocs.io/en/latest/

Table 6
Pros and cons of defense methods for drift attacks

	RAID [52]
(+)	(i) prevents targeted ID attacks (ii) operates independently of ML algorithms/voltage features
(–)	(i) makes use of extended IDs, increases bus load and delays (ii) high-quality PRNGs may have increased costs (iii) the PRNG (seed) must be synchronized across all nodes
	Cryptographic Authentication [53]
(+)	(i) randomizes the ID, prevents targeted attacks on ID or sender (iii) operates independently of ML algorithms/voltage features (ii) offers cryptographic authentication as an additional guarantee (iv) no increase in the bus load
(–)	(i) computationally intensive due to crypto, prone to latencies (ii) requires cryptographic key management
	Boundary estimation
(+)	(i) doesn't require cryptographic operations or PRNGs (ii) doesn't require any change related to frame identifiers (iii) simple implementation that is suitable for DSP deployment
(–)	(i) cannot detect attacks under the voltage threshold (ii) effectiveness depends on classifier, needs experimental evidence

Let us separate between the cold data, which is the data collected when the car has just started and hot data, the data collected after the car has been running for a while. Since the core concept behind the NC classifier is storing only the features describing the centroid of each class in the latent space, we do not need to perform a full retraining on hot data. Instead, for each class, we keep track of the number of samples used for training on cold data and evaluation on hot data, respectively. Then, we calculate the mean point of the hot data, i.e., the supposed centroid that is representative of these never before seen samples. Lastly, we update the centroids for all ECUs by computing a weighted average of the existing (cold) data centroid and the new (hot) data centroid, with the number of cold and hot samples serving as respective weights. For class c, computing the updated centroid can be formalized as such:

$$\mathbb{C}_c^{(\text{updated})} = \frac{n_{\text{c}}^{(cold)} \cdot \mathbb{C}_c^{(\text{cold})} + n_{\text{c}}^{(hot)} \cdot \mathbb{C}_c^{(\text{hot})}}{n_{\text{c}}^{(cold)} + n_{\text{c}}^{(hot)}},$$

where $\mathbb{C}_c^{(\text{updated})}$ is the updated centroid, $n_{\text{c}}^{(cold)}$ is the number of cold training samples, $n_{\text{c}}^{(hot)}$ is number of hot training samples, $\mathbb{C}_c^{(\text{cold})}$ is the original centroid obtained through training on cold data and $\mathbb{C}_c^{(\text{hot})}$ is the centroid computed on hot samples.

It is worth noting one potential shortcoming when using voltage-based schemes are drift attacks on the updating mechanism. The authors of [52] have demonstrated voltage corruption attacks during the collection of voltage samples for retraining and showed that these become effective as impersonation attacks (on the legitimate node). Our solution is no exception to such attacks, but there are at least three ways to defend it. Firstly, the original proposal from [52] can be used, a defense method called RAID (Randomized Identifier Defense), which implies extending the ID of CAN frames to 29-bit and randomizing the additional 18 bits for retraining. Secondly, an even better protection can be achieved by integrating cryptographic material into the ID or data-field of the frames that are used for retraining, which will certify that the message indeed comes from the sender and will randomize the sampled bits as well [53]. Thirdly, a thresholding mechanism (boundary estimation) may be employed. Such a threshold can be derived from the cluster intra-distances that we in fact provided in the supplementary material (Appendix B). Note that in the previous section we already stated that a proximity heuristic holds between the samples and their clusters: $d^c_{\text{min}} > \mu_c + 2\sigma_c$. Therefore $\mu_c + 2\sigma_c$ can be used as a bound for accepting valid samples for the updates. Table 6 depicts the pros and cons of the previously proposed defense methods, focusing on the trade-off between security, computational cost and bus load. Briefly, both RAID [52] and cryptographic authentication [53] provide ID randomization, but require additional computational resources. Moreover, according to the authors [52], RAID increases the bus load by 13% and adds delays of 50 μs by using extended IDs. The synchronization of the PRNG across all nodes, which necessitates using the same seed on each node, can be challenging, as it effectively amounts to managing cryptographic keys. On the other hand, cryptographic authentication of the ID [53] leverages ID authentication by means of dedicated cryptographic functions. Regarding boundary estimation, although the method is less computationally intensive, it remains prone to changes that fall beneath the predetermined threshold and it depends on the way in which the classifier interprets the voltage features. Assessing the effectiveness of this method and determining the exact thresholds may be the subject of future work for us.

The main ECUPrint dataset contains stationary data collected right after the engine was turned on, which we refer to as cold data. It also contains additional data collected in different environmental conditions for two vehicles: Honda Civic and Ford Fiesta, which we call hot data. For the Honda Civic, it includes data collected 10, 30 and 60 minutes after the engine was first started, while the vehicle was moving. For the Ford Fiesta, it includes stationary data collected 30 and 60 minutes after the engine was turned on.

We study the effects of these environmental changes on fingerprinting models trained using cold data. First, we train NC models on cold data for both Honda Civic and Ford Fiesta, with the number of randomly sampled training bits ranging from 10 to 20 for each ECU. Lastly, we evaluate the model on the same number of samples randomly selected from environmental-affected data. Then, we repeat the same evaluation, this time updating the centroid to better fit the hot data. In order to get stable results, the evaluation process was averaged over 1,000 runs.

In Table 7, we compare results obtained for both cars, with and without centroid update, in all available environmental conditions using 20 cold training samples and 20 hot samples for updating the centroids. For the Honda Civic, we first note that environmental changes do not appear to have a major impact after only 10 minutes. Then, most noticeably, after 30 minutes of the engine running, for ECU

Table 7
Results with NC for Honda Civic and Ford Fiesta in various environmental conditions: w/o and with centroid update (20 training samples)

Car	ECU	TAR_C		FRR_C		FAR_C		PRE_C	
		w/o update	update	w/o update	update	w/o update	update	w/o update	update
Honda Civic (10 min)	ECU 1	100%	100%	0%	0%	0.01%	0%	99.97%	100%
	ECU 2	99.99%	100%	0.01%	0%	0.97%	0.01%	94.79%	99.98%
	ECU 3	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 4	97.69%	99.99%	2.31%	0.01%	0%	0%	100%	100%
	ECU 5	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 6	100%	100%	0%	0%	0%	0%	100%	100%
Honda Civic (30 min)	ECU 1	100%	100%	0%	0%	0.01%	0%	99.99%	100%
	ECU 2	99.99%	100%	0.01%	0%	0.01%	0%	93.99%	100%
	ECU 3	76.38%	100%	23.62%	0%	0%	0%	100%	100%
	ECU 4	97.90%	100%	2.10%	0%	0%	0%	100%	100%
	ECU 5	100%	100%	0%	0%	1.58%	0%	96.82%	100%
	ECU 6	100%	100%	0%	0%	0%	0%	100%	100%
Honda Civic (60 min)	ECU 1	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 2	100%	100%	0%	0%	0.32%	0%	98.49%	100%
	ECU 3	1.00%	100%	99.00%	0%	0%	0%	59.34%	100%
	ECU 4	99.37%	100%	0.63%	0%	0%	0%	100%	100%
	ECU 5	100%	100%	0%	0%	4.05%	0%	91.05%	100%
	ECU 6	100%	100%	0%	0%	0%	0%	100%	100%
Ford Fiesta (30 min)	ECU 1	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 2	99.24%	99.94%	0.76%	0.06%	0%	0%	100%	100%
	ECU 3	100%	100%	0%	0%	0.35%	0.03%	73.70%	95.48%
	ECU 4	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 5	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 6	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 7	100%	100%	0%	0%	0%	0%	100%	100%
Ford Fiesta (60 min)	ECU 1	99.99%	100%	0.01%	0%	0.01%	0%	100%	100%
	ECU 2	97.76%	99.93%	2.24%	0.07%	0.01%	0%	99.99%	100%
	ECU 3	100%	100%	0%	0%	0.87%	0.03%	63.68%	96.49%
	ECU 4	100%	100%	0%	0%	0.01%	0%	99.86%	100%
	ECU 5	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 6	100%	100%	0%	0%	0%	0%	100%	100%
	ECU 7	91.62%	100%	8.38%	0%	0%	0%	100%	100%

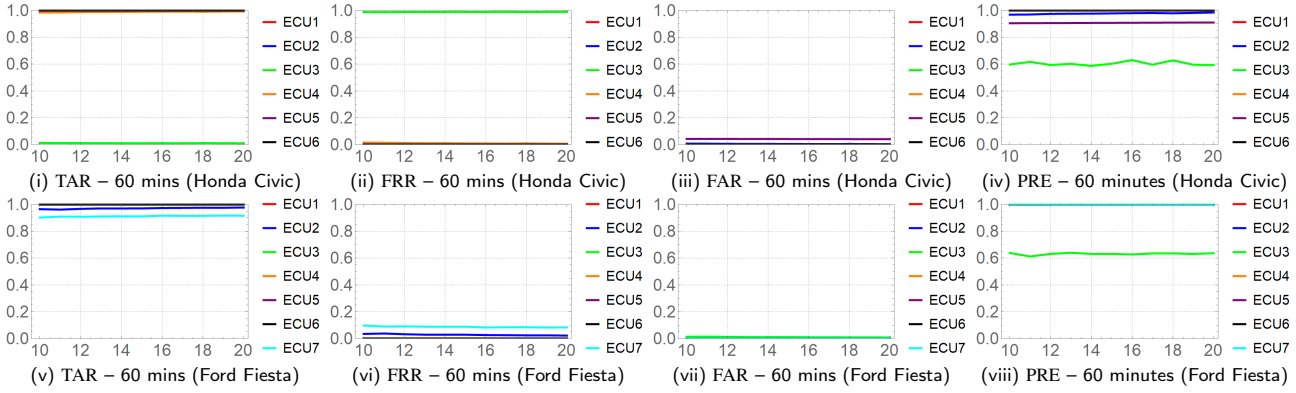
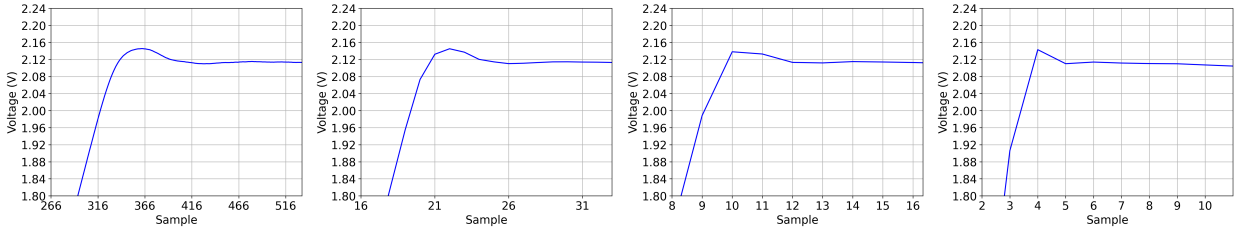


Figure 12: Results without update for Honda Civic and Ford Fiesta at 60 mins after engine start (for 10-20 training samples)

3, TAR increases from 76.38% without update to 100% when updating the centroid. Similarly, FRR decreases from 23.62% to 0%. Therefore, by updating the centroid, FAR decreases from 1.58% to 0%, while PRE increases from 96.82% to 100%. After 60 minutes, updating the centroids increases TAR from 1% to 100% and reduces FRR from 99% to 0%. Consequently, it appears most samples from ECU 3 were originally misplaced inside ECU 5, with a FAR value of 4.05% and a PRE value of 91.05%. After updating, these metrics become 0% and 100%, respectively. As for the Ford Fiesta, after 60 minutes, TAR increases for ECU 7 from 91.62% to 100% and FRR decreases from 8.38% to 0%. After employing the update mechanism, we report a maximum of 0.03% FAR across the other ECUs, with precision between 96.49% and 100%. We note that for both vehicles, there are some ECUs where the reported metrics without update are very close to the optimum, e.g., between 97.69% and 99.99% for TAR, which have also improved after updating. In some cases, e.g., ECU3 for Ford Fiesta after 30 and 60 minutes, the metrics have improved up to a value very close to 100%. We exemplify for the 60 minutes case: FAR decreases from 0.87% to 0.03% and PRE increases from 63.68% to 96.49%.



(i) Original sample, 500 MHz (ii) 16× downsampling, 31.25 MHz (iii) 32× downsampling, 15.62 MHz (iv) 64× downsampling, 7.81 MHZ

Figure 13: Example of downsampled data on the rising edge from a Ford Fiesta sample (ID 04A)

We further analyze the effect of the update mechanism by plotting the average evaluation metrics with respect to the number of samples used during training for each ECU. For brevity, in Figure 12, we display these results only after 60 minutes of driving for both vehicles, sub-figures (i), (ii), (iii) and (iv) show the trends for TAR, FRR, FAR and PRE on the Honda Civic. While most ECUs are not significantly affected by environmental changes, for ECU 3, TAR remains very close to 0% while FRR is close to 100% no matter the number of training samples used. We note that FAR is 0%, PRE is around 60%. Notably, in some specific corner cases, PRE can be a misleading metric. For instance, this average of 60% stems from two distinct situations: either there are very few true positives detected, e.g. 4, and there are no false positives or there are no true positives or false positives detected at all. This translates to PRE being 100% despite very few correct detections as there are no false positives, or PRE being 0% due to a division by 0, which is handled in the standard way of using 0% as an estimate for it. In other words, PRE can sometimes be very high because there are no falsely classified samples, but that does not automatically signal a good detection rate. Instead, TAR becomes more relevant as it depicts the percentage of correctly detected samples. Meanwhile, Figure 12, sub-figures (v), (vi), (vii) and (viii) show the trends of the metrics on stationary data collected from the Ford Fiesta after 30 and 60 minutes with the engine running. While for most ECUs, the metrics remain invariant to the number of training samples, with values close to their respective optimum, ECU 7 remains the most far off by approximately 10% in the case of FRR and FAR no matter how many training samples are used, while FAR and PRE exhibit a similar behavior to that from 60 minutes. Based on the difference in behavior after 60 minutes for both vehicles, we conclude that the centroid update mechanism is necessary to construct a voltage fingerprinting algorithm capable of reporting close to 100% correct identification.

6. Sampling Rate and the Impact of Noise

In this section, we address two relevant issues: the impact of the sampling rate, which directly influences the deployability of the solution, and the influence of noise, which is commonly present in in-vehicle networks where a large number of electric and electromechanical components coexist.

6.1. Ablation Study on Sampling Rate

Our system is made up of two distinct components: a) the unsupervised clustering algorithm which is required to provide a reliable GT and b) the supervised Nearest Centroid based IDS that works with the centroids of the clusters provided by the first component. Regarding the first component, reducing the dimensions of the PCA transform to 2, 4, 8 led to unsatisfactory results: HDBSCAN with PCA-2 failed on Ford Kuga and with PCA-4, PCA-8, it failed on Hyundai ix35. Since clustering can be done off-line, i.e., there are no real-time constraints, while 16 components mean a mere 128 Bytes of memory per ECU, we consider that going lower is not necessary and settle on PCA-16. As such, we focus on the Nearest Centroid detection scheme next, conducting an ablation study on the behavior of the classifier when the signals are downsampled.

Concretely, the dataset used in this work contains voltage samples collected at 500 MHz with a Pico Scope 5000 Series. The Pico Scope has two key components used to sample voltage data, the XC6SLX25 FPGA from Xilinx and ADC08D502 module from Texas Instruments with 500 milion samples per second (MSPS) capability. As also stated in [5], both are designed for automotive use with a price close to 100$ so, in the future, a similar hardware configuration may be considered for in-vehicle high sampling rate voltage based applications. Therefore, another valid concern is related to the fact that this may be too demanding for deployment inside vehicles today due to the high sampling rate. This equates to evaluating whether the Nearest Centroid detection scheme is resilient to downsampling the data to lower frequencies and determining a minimal frequency requirement. Given that one example in our dataset (i.e., one bit) has 1600 samples, we begin by decimating the signals by keeping every 16th, 32nd and 64th sample respectively. In other words, we downsample the signals from 500 MHz to 31.25 MHz, 15.62 MHz and 7.81 MHz. Figure 13 (i) shows the rising edge of a raw bit, sampled at 500 MHz from ID 04A of Ford Fiesta, while Figure 13 (ii) shows the same rising edge downsampled to 31.25 MHz. Then, Figure 13 (iii) shows the rising edge downsampled to 15.62 MHz and Figure 13 (iv) displays it downsampled to 7.81 MHz. The degradation of the signal is especially visible as the sampling frequency lowers.

We note that downsampling may also affect temporal alignment, which is critical for the detection rate – an aspect that appears to have been overlooked in related works.

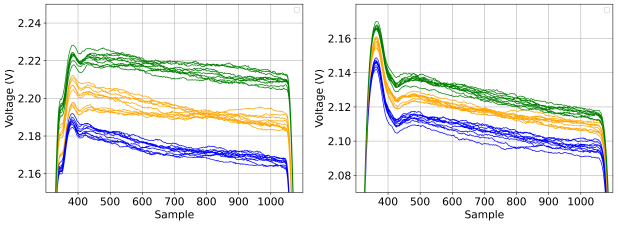
Two distinct acquisition scenarios can be envisioned: (i) an asynchronous ADC clock, which samples at a lower rate and whose phase relative to the bit transitions is random, and (ii) a triggered acquisition, which begins when the input signal crosses a predefined threshold. In what follows, we present results for both scenarios (asynchronous and triggered) showing that alignment is even more critical than the sampling rate itself. The triggered scenario remains extremely stable despite signal decimation, whereas the asynchronous one exhibits phase variability: for example, when the signal is decimated by a factor of 16, sampling may begin at a random phase between 0 and 15, while in the triggered case it consistently starts at the rising edge (set to 0.25V in our experiments).

Table 8 shows results with NC on cold data from Honda Civic and Ford Fiesta for both asynchronous and triggered downsampling. For brevity, we show only TAR and FAR, encompassing both false negatives and false positives. On one hand, detection rate degradation is quite visible when downsampling asynchronously. When the decimation factor is set to 16, 3 ECUs suffer from degradation on the Honda Civic, with TAR as low as 84.16% and FAR as high as 5.26%, while on the Ford Fiesta, the lowest reported TAR value is 79.83% and the highest FAR is 5.76%, only one ECU remaining unaffected. Degradation intensifies when the decimation factor is increased to 32, with TAR as low as 58.10% and FAR as high as 12.09% on the Honda Civic, with all ECUs affected. On the Ford Fiesta, we report TAR as low as 51.23% and FAR as high as 12.14%, with one ECU unaffected. Performance loss increases even more when using a decimation factor of 64. For Honda Civic, the lowest TAR reported is 36.48% while the highest FAR is 11.84%, while on Ford Fiesta, TAR is as low as 32.73% and FAR is as high as 13.08%. One ECU remains mostly unaffected, with a TAR decrease of 1.1%. On the other hand, triggered downsampling shows minimal degradation from the results reported in Table 5. On Honda Civic, TAR drops for ECU 2 by a minimum of 0.03% at 15.62 MHz and a maximum of 0.47% at 7.81 MHz, with FAR increasing only by 0.1% at the lowest frequency. Meanwhile, on the Ford Fiesta, no degradation is noticeable.

We conclude that the effect of downsampling is highly dependent on whether the procedure is performed asynchronous or triggered, with the asynchronous downsampling suffering from increased performance degradation due to the likelihood of missing the beginning of samples. The triggered downsampling performed excellently for the two cars, but it should not be interpreted in a very positive way since the ECUs in the two cars are sufficiently far from each other as shown in Appendix B from the supplementary material. Depending on the distance between ECUs, higher sampling rates may be needed.

6.2. Impact of Electrical Noise

Noise is indeed a major concern for in-vehicle voltage data. We note that the CAN bus is particularly resilient to electrical noise due to its differential wiring. However, it is not immune to noise and the voltage data that we used had a number of relevant voltage fluctuations. To further clarify this point, we discuss the voltage patterns from several samples in the dataset. Concretely, in Figure 14 (i) and (ii), we show 30 bits from ID 156 of Honda Civic and ID 04A of Ford Fiesta, respectively. The bits in the middle (orange) represent samples that are closest to the mean of the particular ID, while the upper samples (green) and the lower samples (blue) represent the samples that are furthest from the mean in the positive or negative directions. It can be clearly seen that, for ID 156, voltage levels on the plateau vary by 70mV while for ID 04A, plateau voltage levels also vary by 70mV. The true acceptance rates of the NC classifier were 100% on samples from ECU 6 of Honda Civic and ECU 1 of Ford Fiesta (that include these particular bits) showing that the classifier performance remains stable under such noise.



(i) Rising edge, plateau and falling edge of ID 156 from Honda Civic

(ii) Rising edge, plateau and falling edge of ID 04A from Ford Fiesta

Figure 14: Examples of noise-affected voltage data

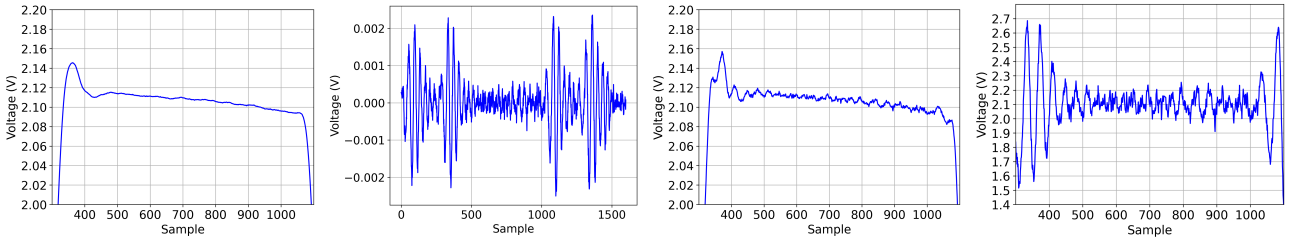
We have also endeavored to test the resilience of the scheme on various signal-to-noise-ratios (SNRs). For this purpose, we used a high-pass filter to collect noise from the samples, then amplify and add it to existing bits. In this sense, we employed Python's *Scipy*[5] library to construct a 5th order high-pass filter with a cut-off frequency of 10 MHz. Figure 15 (i) shows the original shape of the signal for one sample from ID 04A of Ford Fiesta, focusing specifically on the plateau area, while Figure 15 (ii) displays the noise extracted from this sample having a SNR of approximately 67dB. Figure 15 (iii) shows an augmented version of the sample, when the extracted noise is amplified to reach a SNR of approximately 48dB. Then, Figure 15 (iv) shows the same augmentation at a SNR of approximately 18 dB. Extrapolating at the vehicle level, we then apply noise to Honda Civic and Ford Fiesta cold data so the average SNR is approximately 48dB and 18dB respectively.

Concretely, we train NC models on 20 randomly sampled raw bits from each ECU, and apply synthetic noise only to the testing samples. PCA-16 is still used for dimensionality reduction, and we average out our results over 100 iterations, different training samples being randomly selected on each run. For an approximate average of 48dB SNR across all samples, the results are similar to those obtained when evaluating on raw data for both vehicles, with negligible differences of 0.01% that can be attributed to the random sampling of data across multiple runs. When the noise is

[5] https://scipy.org/

Table 8
Results with NC for Honda Civic and Ford Fiesta on downsampled data (20 training samples)

Car	ECU	16 × decim. async		32 × decim. async		64 × decim. async		16 × decim. triggered		32 × decim. triggered		64 × decim. triggered	
		TAR_C	FAR_C	TAR_C	FAR_C	TAR_C	FAR_C	TAR_C	FAR_C	TAR_C	FAR_C	TAR_C	FAR_C
Honda Civic	ECU 1	100%	0%	99.99%	0%	93.03%	1.03%	100%	0%	100%	0%	100%	0%
	ECU 2	84.16%	5.26%	58.10%	12.09%	36.48%	11.84%	99.95%	0%	99.96%	0%	99.52%	0%
	ECU 3	99.99%	0.01%	86.14%	3.09%	57.32%	11.28%	100%	0%	100%	0%	100%	0%
	ECU 4	84.86%	3.66%	64.20%	7.00%	47.89%	10.35%	100%	0.01%	100%	0.01	100%	0.11%
	ECU 5	100%	0%	97.98%	0.40%	78.06%	4.87%	100%	0%	100%	0%	100%	0%
	ECU 6	100%	0%	97.80%	0.79%	77.45%	7.07%	100%	0%	100%	0%	100%	0%
Ford Fiesta	ECU 1	96.83%	1.25%	77.68%	5.05%	47.74%	6.52%	100%	0%	100%	0%	100%	0%
	ECU 2	82.80%	0.29%	56.56%	1.07%	37.67%	4.10%	99.99%	0%	99.99%	0%	99.99%	0%
	ECU 3	86.75%	5.76%	61.39%	12.14%	39.45%	17.16%	100%	0.01%	100%	0.01%	100%	0.01%
	ECU 4	84.85%	0.09%	68.23%	4.10%	45.49%	13.01%	100%	0%	100%	0%	100%	0%
	ECU 5	97.15%	0%	73.59%	0.26%	47.99%	4.08%	100%	0%	100%	0%	100%	0%
	ECU 6	100%	0%	100%	0%	98.90%	0%	100%	0%	100%	0%	100%	0%
	ECU 7	79.83%	2.22%	51.23%	10.53%	32.73%	13.08%	100%	0%	100%	0%	100%	0%



(i) Original data sample (ii) ~ 67dB SNR (iii) ~ 48dB SNR (iv) ~ 18dB SNR

Figure 15: Example of noise being amplified on voltage data (Ford Fiesta, ID 04A)

increased, averaging out to approximately 18dB SNR across all samples, we notice a decrease in TAR of 3% for ECU 4 and a decrease in PRE of 5% for ECU 2. For the Ford Fiesta, we observe a decrease in PRE of 1.7% for ECU 3 and a decrease in TAR of 0.04% for ECU 2. This shows that the Nearest Centroid detection scheme performs well even when the SNR reaches an average of 18dB.

Last but not least, some concerns can be expressed for the fact that the car battery may exhibit changes in the state of charge as the vehicle is running and especially at vehicle start-up. However, CAN bus transceivers are not powered at 12V directly from the battery. Instead, they are typically powered by 5V supplied through a voltage regulator and operate within a predefined range, for example, between 4.5V and 5.5V [54]. Thus, the transceiver expects the voltage regulator to deliver voltage in that range and has its own protection mechanisms (the popular CAN bus transceiver TJA1042 will disconnect when voltage drops below the acceptable threshold). Any fluctuations in the vehicle's battery voltage (such as those occurring during engine cranking, regenerative braking, or alternator charging) are expected to be absorbed and regulated by the voltage regulator. Indeed, our dataset covers only 10, 30, and 60 minutes post-engine start, and this is expected to cover substantial voltage-related events. Regarding a full coverage, even extending the dataset to several hours or days would not guarantee capturing all possible electrical states of a vehicle. Still, if any events occur, they would likewise be mitigated by the 5V regulator. Most short-term variations on the supply line arise from normal electrical activity in the vehicle (e.g., pumps, relays, switching loads) and from temperature-induced changes in component resistance. The fluctuations observed after 10, 30, and 60 minutes of operation are small but meaningful – yet, they can be effectively compensated through retraining, as previously demonstrated.

7. Conclusion

The specifics of voltage-based fingerprints make this kind of data well-suited for the HDBSCAN clustering algorithm. Still, we show that without proper parametrization, this algorithm fails for many of the vehicles, a reason for which careful tuning of the parameters is necessary. In particular, the value of ϵ seems critical, as we are able to identify three additional ECUs while also eliminating another. Once the clusters are fixed, the road is open for an efficient classifier like the Nearest Centroid for message source identification, through the use of centroids as ECU-level fingerprints. We show that the detection scheme works well up to a 18 dB SNR and, through an ablation study, we demonstrate that it can also cope to downsampling as long as a trigger is used to align the samples. Also, we provide a very efficient way of updating these fingerprints, which is critical for realistic deployment, since in-vehicle voltage fingerprints are susceptible to changes over time. Concretely, we obtain close to 99.93% TAR after 60 minutes of driving, with only 264 Bytes of RAM required per centroid (including Python-specific metadata). This demonstrates a 4–20 times reduction in memory usage relative to widely used algorithms like SVM and RF and proves that more attention

should be given to this simple and efficient classifier when dealing with voltage data as more complex solutions may lead to unnecessary complications.

Acknowledgment

This research is partially supported by the project "Romanian Hub for Artificial Intelligence - HRIA", Smart Growth, Digitization and Financial Instruments Program, 2021-2027, MySMIS no. 334906.

Appendix A. Experimental Data Regarding the Failures of K-means

The tables from this appendix include the experimental data for some failure cases observed in the K-means clustering. Concretely, it contains the most relevant performance metrics for three vehicles in our dataset: Table 9 reports the results for the Ford Kuga, Table 10 for the Hyundai ix35 and Table 11 for the Ford Fiesta. Each table includes the GT ECU labels and their IDs, the number of samples in the GT, the number of samples reported by K-Means, as well as TAR_C, FRR_C, FAR_C and PRE_C. A discussion on these results can be found in the main body of the work.

Appendix B. Additional Experimental Results

The tables from this appendix present the detailed performance metrics for the clustering results across all vehicles in the dataset. Some technical comments are provided in the experimental results subsection of the main paper, here we simply summarize the numerical results.

Each table header lists the ground-truth (GT) ECU label with its corresponding IDs, the number of GT samples for each ECU, and the number of samples our clustering assigns correctly. It also includes the metrics defined in the main body of the paper: true acceptance rate (TAR_C), false rejection rate (FRR_C), false acceptance rate (FAR_C), precision (PRE_C), intra-cluster mean distance (μ_c), intra-cluster standard deviation (σ_c), and the distance to the nearest centroid outside the cluster (d^c_{min}).

Briefly, Table 12 reports results for the Dacia Duster, Table 13 for the Opel Corsa and Table 14 for the Ford Ecosport. Table 15 presents the Dacia Logan results, and Table 16 those for the Honda Civic. Tables 17 and 18 summarize the Hyundai i20 and Hyundai ix35 results, while Table 19 covers the Ford Fiesta. Results for the Ford Kuga appear in Table 20, and Table 21 contains the results for the John Deere tractor.

CRediT authorship contribution statement

Patricia Iosif: Software, writing - original draft preparation, investigation, experimentation, writing - review and editing. **Lucian Popa:** Data sanitation, software, investigation, writing - review and editing. **Bogdan Groza:** Conceptualization of this study, methodology and supervision, writing - review and editing, project administration.

References

[1] S. Nie, L. Liu, Y. Du, Free-fall: Hacking tesla from wireless to can bus, Briefing, Black Hat USA 25 (2017) 16.

[2] J. Jepson, R. Chatterjee, J. Daily, Commercial vehicle electronic logging device security: Unmasking the risk of truck-to-truck cyber worms, in: Symposium on Vehicles Security and Privacy (VehicleSec), volume 2024, 2024, pp. 1–12.

[3] X. Zhang, Y. Tu, Y. Long, L. Shan, M. A. Elsaadani, K. Fu, Z. Lin, X. Hei, From virtual touch to tesla command: Unlocking unauthenticated control chains from smart glasses for vehicle takeover, in: 2024 IEEE Symposium on Security and Privacy (SP), IEEE, 2024, pp. 2366–2384.

[4] ISO, ISO11898-1. Road vehicles — Controller Area Network (CAN) — Part 1: Data link layer and physical coding sublayer, Standard, 3rd edition, International Organization for Standardization, 2024.

[5] L. Popa, B. Groza, C. Jichici, P.-S. Murvay, Ecuprint—physical fingerprinting electronic control units on can buses inside cars and sae j1939 compliant vehicles, IEEE Transactions on Information Forensics and Security 17 (2022) 1185–1200.

[6] G. Zhang, Y. Li, Voltage inspector: Sender identification for in-vehicle can bus using voltage slice, Computers & Security 145 (2024) 104017.

[7] ISO, ISO11898-2. Road vehicles — Controller Area Network (CAN) — Part 2: High-speed physical medium attachment (PMA) sublayer, Standard, 2nd edition, International Organization for Standardization, 2024.

[8] SAE, High-Speed CAN (HSC) for Vehicle Applications at 500 KBPS, Standard, SAE International, 2022.

[9] H. M. Song, J. Woo, H. K. Kim, In-vehicle network intrusion detection using deep convolutional neural network, Vehicular Communications 21 (2020) 100198.

[10] Y. Du, Y. Li, P. Cheng, Z. Han, Y. Wang, Ugl: A comprehensive hybrid model integrating gcn and lstm for enhanced intrusion detection in uav controller area networks, Computer Networks 262 (2025) 111157.

[11] P. Dini, E. Soldaini, S. Saponara, Design and test of an embedded real-time & compact voltage fingerprinting algorithm for enhanced automotive cybersecurity, IEEE Access (2025).

[12] P. Dini, M. Zappavigna, E. Soldaini, S. Saponara, Embedded machine learning-based voltage fingerprinting for automotive cybersecurity, IEEE Access (2025).

[13] S.-C. Liu, C.-M. Wu, Fpga implementation for bayesian gaussian mixture model-based ecu identification on can buses, in: 2024 10th International Conference on Systems and Informatics (ICSAI), IEEE, 2024, pp. 1–5.

[14] Z. Deng, J. Liu, Y. Xun, J. Qin, Identifierids: A practical voltage-based intrusion detection system for real in-vehicle networks, IEEE Transactions on Information Forensics and Security 19 (2023) 661–676.

[15] N. Liu, C. Moreno, M. Dunne, S. Fischmeister, vprofile: Voltage-based anomaly detection in controller area networks, in: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2021, pp. 1142–1147.

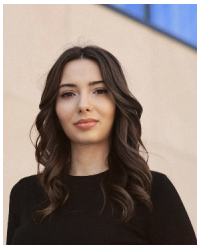
[16] J. Shi, Z. Xie, L. Dong, X. Jiang, X. Jin, Ids-dec: A novel intrusion detection for can bus traffic based on deep embedded clustering, Vehicular Communications 49 (2024) 100830.

[17] Y. Xin, X. Wang, L. Lu, S. Zhuo, Y. Jiang, A. K. Singh, K. Ren, M. Yang, K. Wu, Luft-can: A lightweight unsupervised learning based intrusion detection system with frequency-time analysis for vehicular can bus, Journal of Systems Architecture (2025) 103567.

[18] P.-S. Murvay, B. Groza, Source identification using signal characteristics in controller area networks, IEEE Signal Processing Letters 21 (2014) 395–399.

[19] J. Li, M. Zhang, Y. Lai, A light-weighted machine learning based ecu identification for automotive can security, in: 2023 International Conference on Networking and Network Applications (NaNA), IEEE, 2023, pp. 545–550.

[20] G. Baldini, Voltage based electronic control unit (ecu) identification with convolutional neural networks and walsh–hadamard transform, Electronics 12 (2022) 199.
[21] S. Gehrer, J. Guajardo, Phidias: Power signature host-based intrusion detection in automotive microcontrollers, in: Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security, 2024, pp. 36–47.
[22] K.-T. Cho, K. G. Shin, Viden: Attacker identification on in-vehicle networks, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1109–1123.
[23] W. Choi, K. Joo, H. J. Jo, M. C. Park, D. H. Lee, Voltageids: Low-level communication characteristics for automotive intrusion detection system, IEEE Transactions on Information Forensics and Security 13 (2018) 2114–2129.
[24] J. Zhou, P. Joshi, H. Zeng, R. Li, Btmonitor: Bit-time-based intrusion detection and attacker identification in controller area network, ACM Transactions on Embedded Computing Systems (TECS) 18 (2019) 1–23.
[25] Y. Wei, C. Cheng, G. Xie, OFIDS: online learning-enabled and fingerprint-based intrusion detection system in controller area networks, IEEE Transactions on Dependable and Secure Computing 20 (2022) 4607–4620.
[26] H. Wei, Q. Ai, Y. Zhai, Y. Zhang, Automotive security: Threat forewarning and ecu source mapping derived from physical features of network signals, IEEE Transactions on Intelligent Transportation Systems 25 (2023) 2479–2491.
[27] M. Zhang, J. Li, Y. Lai, S. Huan, W. Shang, A lightweight voltage-based ecu fingerprint intrusion detection system for in-vehicle can bus, IEEE Transactions on Vehicular Technology (2025).
[28] A. McEntarffer, L. Zhang, Y. W. Liu, M. Pesé, Towards a comprehensive evaluation of voltage-based fingerprinting for the can bus, in: Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing, 2025, pp. 1886–1893.
[29] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, D. H. Lee, Identifying ecus using inimitable characteristics of signals in controller area networks, IEEE Transactions on Vehicular Technology 67 (2018) 4757–4770.
[30] M. Kneib, C. Huth, Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks, in: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, 2018, pp. 787–800.
[31] M. Kneib, O. Schell, C. Huth, On the robustness of signal characteristic-based sender identification, arXiv preprint arXiv:1911.09881 (2019).
[32] M. Kneib, O. Schell, C. Huth, Easi: Edge-based sender identification on resource-constrained platforms for automotive networks., in: NDSS, 2020, pp. 1–16.
[33] T. Xu, X. Lu, L. Xiao, Y. Tang, H. Dai, Voltage based authentication for controller area networks with reinforcement learning, in: ICC 2019-2019 IEEE International Conference on Communications (ICC), IEEE, 2019, pp. 1–5.
[34] L. Xiao, X. Lu, T. Xu, W. Zhuang, H. Dai, Reinforcement learning-based physical-layer authentication for controller area networks, IEEE Transactions on Information Forensics and Security 16 (2021) 2535–2547.
[35] O. Schell, M. Kneib, Valid: Voltage-based lightweight intrusion detection for the controller area network, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, 2020, pp. 225–232.
[36] M. Rogers, P. Weigand, J. Happa, K. Rasmussen, Detecting can attacks on j1939 and nmea 2000 networks, IEEE Transactions on Dependable and Secure Computing 20 (2022) 2406–2420.
[37] Y. Xun, Z. Deng, J. Liu, Y. Zhao, Side channel analysis: A novel intrusion detection system based on vehicle voltage signals, IEEE Transactions on Vehicular Technology 72 (2023) 7240–7250.
[38] Y. Xun, Y. Zhao, J. Liu, Vehicleeids: A novel external intrusion detection system based on vehicle voltage signals, IEEE Internet of Things Journal 9 (2021) 2124–2133.
[39] L. Yin, J. Xu, C. Wang, Q. Wang, F. Zhou, Detecting can overlapped voltage attacks with an improved voltage-based in-vehicle intrusion detection system, Journal of Systems Architecture 143 (2023) 102957.
[40] M. Foruhandeh, Y. Man, R. Gerdes, M. Li, T. Chantem, Simple: Single-frame based physical layer identification for intrusion detection and prevention on in-vehicle networks, in: Proceedings of the 35th annual computer security applications conference, 2019, pp. 229–244.
[41] C. Ricardo J.G.B, D. Moulavi, J. Sander, Density-based clustering based on hierarchical density estimates, in: Advances in Knowledge Discovery and Data Mining, PAKDD, 2013, pp. 160–172.
[42] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, AAAI Press, 1996, p. 226–231.
[43] R. C. Prim, Shortest connection networks and some generalizations, The Bell System Technical Journal 36 (1957) 1389–1401.
[44] comma.ai, Opendbc: Open-source vehicle database, GitHub repository, 2025. URL: https://github.com/commaai/opendbc, accessed: 2025-12-07.
[45] D. Murzinov, awesome-automotive-can-id, GitHub repository, 2025. URL: https://github.com/iDoka/awesome-automotive-can-id, accessed: 2025-12-07.
[46] R. Thorndike, Who belongs in the family?, Psychometrika 18 (1953) 267–276.
[47] E. Schubert, J. Sander, M. Ester, H.-P. Kriegel, X. Xu, Dbscan revisited, revisited, ACM Transactions on Database Systems (TODS) 42 (2017) 1 – 21.
[48] E. B. Fowlkes, C. L. Mallows, A method for comparing two hierarchical clusterings, Journal of the American statistical association 78 (1983) 553–569.
[49] D. . Company, Operator's manual ompfp13250: Canbus theory of operation, http://manuals.deere.com/omview/OMPFP13250_19/RW00482,000018D_19_20130516.html, 2013. Accessed: 2026-01-26.
[50] FixMyCarInfo.com, Hyundai tucson / ix35 2 (lm) (2012) – wiring diagrams, 2025. URL: https://fixmycarinfo.com/hyundai/tucson-ix35/hyundai-tucson-ix35-2-lm-2012-wiring-diagrams/.
[51] J. J. Rocchio Jr, Relevance feedback in information retrieval, The SMART retrieval system: experiments in automatic document processing (1971).
[52] R. Bhatia, V. Kumar, K. Serag, Z. B. Celik, M. Payer, D. Xu, Evading voltage-based intrusion detection on automotive can., in: NDSS, 2021, pp. 1–17.
[53] S. Woo, D. Moon, T.-Y. Youn, Y. Lee, Y. Kim, Can id shuffling technique (cist): Moving target defense strategy for protecting in-vehicle can, IEEE Access 7 (2019) 15521–15536.
[54] NXP Semiconductors, TJA1042: High-speed CAN Transceiver with Standby Mode, rev. 11 ed., 2023. URL: https://www.nxp.com/docs/en/data-sheet/TJA1042.pdf, product data sheet, 16 January 2023.



Patricia Iosif started her Ph.D. studies in 2022 at Politehnica University of Timisoara (UPT). She graduated B.Sc.in 2020 and M.Sc studies in 2022 at the same university. She works as an applied scientist with a professional background of 4 years. Her research interests are related to the application of artificial intelligence methods for clustering and classification of smart devices and automotive ECUs using physical characteristics such as acceleration and voltage samples.

Lucian Popa received his Ph.D. degree in 2024 at Politehnica University of Timisoara (UPT). He graduated his B.Sc. in 2015 and his M.Sc. studies in 2017 at the same university. He has a 4 year background as a software developer and later as

system engineer in the automotive industry as former employee of Autoliv, Veoneer, Magna, and is currently working for Compal. His research interests are in automotive security with focus on the security of in-vehicle buses.

Bogdan Groza is Professor at Politehnica University of Timisoara (UPT). He received his Ph.D. and Dipl.Ing. degrees from UPT in 2004 and 2008 respectively. In 2016 he successfully defended his habilitation thesis having as core subject the design of cryptographic security for automotive embedded devices and networks. He has been actively involved inside UPT with the development of laboratories by Continental Automotive and Vector Informatik. In addition to regular participation in national and international research projects in information security, he led the CSEAMAN project (2015-2017) and the PRESENCE project (2018-2020), two research programs dedicated to automotive security funded by the Romanian Authority for Scientific Research and Innovation

Table 9
Results with K-means for Ford Kuga (Clusters = 11)

ECU	IDs in GT	Clustered IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C
ECU 1	0B0, 140	0B0, 140	512	512	100%	0%	0%	100%
ECU 2	3E2, 3EA, 455	not seen as a distinct cluster, IDs are clustered within ECU 4	97	-	0%	100%	0%	0%
ECU 3	050, 06A, 0D0, 0E0, 0F0, 0F5, 100, 2B0	050, 06A, 0D0, 0E0, 0F0, 0F5, 100, 2B0	6622	3708	56%	44%	0%	100%
		ECU 3*: 050, 06A, 0D0, 0E0, 0F0, 0F5, 100, 2B0 (part of the samples are seen as a distinct ECU)	6622	2914	0%	0%	100%	N/A
ECU 4	030, 0C8, 150, 17E, 260, 290, 310, 360, 380, 3B4, 400, 405, 40A, 420, 435, 581	030, 0C8, 150, 17E, 260, 290, 310, 360, 380, 3B4, 400, 405, 40A, 420, 435, 581, 3E2, 3EA, 455	3344	3441	100%	0%	0.39%	97.18%
ECU 5	060, 070, 080, 090, 0A0, 0C0, 0F8, 120, 130, 138, 1A0, 1B0, 200, 270, 280, 2D8, 2F0, 340	060, 070, 080, 090, 0A0, 0C0, 0F8, 120, 130, 138, 1A0, 1B0, 200, 270, 280, 2D8, 2F0, 340	8077	4350	53.86%	46.14%	0%	100%
		ECU 5*: 060, 070, 080, 090, 0A0, 0C0, 0F8, 120, 130, 138, 1A0, 1B0, 200, 270, 280, 2D8, 2F0, 340 (part of the samples are seen as a distinct ECU)	8077	3727	0%	0%	100%	N/A
ECU 6	160, 180, 190, 1C0, 1D0, 1E0, 210, 213, 218, 252, 2D0, 2D4	160, 180, 190, 1C0, 1D0, 1E0, 210, 213, 218, 252, 2D0, 2D4	5876	2958	50.34%	49.66%	0%	100%
		ECU 6*: 160, 180, 190, 1C0, 1D0, 1E0, 210, 213, 218, 252, 2D0, 2D4 (part of the samples are seen as a distinct ECU)	5876	2918	0%	0%	100%	N/A%
ECU 7	04A, 04B, 208, 388	not seen as a distinct cluster, IDs are clustered within ECU 9	773	-	0%	100%	0%	0%
ECU 8	2E0	not seen as a distinct cluster, IDs are clustered within ECU 9 and ECU 11	90	-	0%	100%	0%	0%
ECU 9	170, 229, 2A0, 2A5	170, 229, 2A0, 2A5, 04A, 04B, 208, 388, 2E0	924	1783	100%	0%	3.17%	51.82%
ECU 10	010	010	1007	1007	100%	0%	0%	100%
ECU 11	269	269, 2E0	702	706	100%	0.01%	0%	99.43%

* indicates samples misidentified by K-means as a distinct ECU

Table 10
Results with K-means for Hyundai ix35 (Clusters = 6)

ECU	IDs in GT	Clustered IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C
ECU 1	350	350	1461	1461	100%	0%	0%	100%
ECU 2	165, 2B0, 5E4	165, 2B0, 5E4	2342	2342	100%	0%	0%	100%
ECU 3	4F0, 690	not seen as a distinct cluster, IDs are clustered within ECU 4, ECU 4* and ECU 4**	280	-	0%	100%	0%	0%
ECU 4	153, 164, 1F1, 220, 430, 4B1, 4D0	153, 164, 1F1, 220, 430, 4B1, 4D0, 4F0, 690	6210	2679	41.22%	58.78%	0.87%	95.56%
		ECU 4*: 153, 164, 1F1, 220, 430, 4B1, 4D0 (part of the samples are seen as a distinct ECU) and 4F0, 690	6210	2096	0%	0%	100%	N/A
		ECU 4**: 153, 164, 1F1, 220, 430, 4B1, 4D0 (part of the samples are seen as a distinct ECU) and 4F0	6210	1715	0%	0%	100%	N/A
ECU 5	0A0, 0A1, 18F, 260, 2A0, 316, 329, 545	0A0, 0A1, 18F, 260, 2A0, 316, 329, 545, 429, 5A0, 5A1, 5A2	9334	9563	100%	0%	2.18%	97.61%
ECU 6	429, 5A0, 5A1, 5A2	not seen as a distinct cluster, IDs are clustered within ECU 5	229	-	0%	100%	0%	0%
ECU 4*	153, 164, 1F1, 220, 430, 4B1, 4D0	-	2096	-	0%	0%	100%	N/A
ECU 4**	153, 164, 220, 4B1, 4D0	-	1715	-	0%	0%	100%	N/A

*,** indicate samples misidentified by K-means as a distinct ECU

Table 11
Results with K-means for Ford Fiesta (Clusters = 7)

ECU	IDs in GT	Clustered IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C
ECU 1	023, 04A, 04B, 460	023, 04A, 04B, 460	498	498	100%	0%	0%	100%
ECU 2	073, 090, 20E, 20F, 211, 212, 213, 215, 216, 2C3, 4B0	073, 090, 20E, 20F, 211, 212, 213, 215, 216, 4B0	8759	4867	55.57%	44.43%	0%	100%
		ECU 2*: 073, 090, 20E, 20F, 211, 212, 213, 215, 216, 2C3, 4B0 (part of the samples are seen as a distinct ECU) and 150	8759	4100	0%	0%	100%	N/A
ECU 3	150	not seen as a distinct cluster, IDs are clustered within ECU 2*	208	-	0%	100%	0%	0%
ECU 4	190, 275, 2C1, 400, 405, 430, 432, 433, 4E3, 4F2	190, 275, 2C1, 400, 405, 430, 432, 433, 4E3, 4F2	1398	1398	100%	0%	0%	100%
ECU 5	0FD, 200, 201, 203, 205, 228, 231, 232, 261, 268, 280, 2BA, 360, 364, 420, 424, 428, 4F1	0FD, 200, 201, 203, 205, 228, 231, 232, 261, 268, 280, 2BA, 360, 364, 420, 424, 428, 4F1	9458	9458	100%	0%	0%	100%
ECU 6	080, 240	080, 240	1340	1340	100%	0%	0%	100%
ECU 7	455	455	68	68	100%	0%	0%	100%

* indicates samples misidentified by K-means as a distinct ECU

Table 12
Results with HDBSCAN for Dacia Duster ($\epsilon = 0.5$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	161, 181, 1F9, 551, 5DD, 65C	4714	4714	100%	0%	0%	100%	0.179	0.072	11.011
ECU 2	244, 284, 285, 354	2716	2716	100%	0%	0%	100%	0.194	0.104	4.160
ECU 3	1A5	1512	1512	100%	0%	0%	100%	0.192	0.067	4.160

Table 13
Results with HDBSCAN for Opel Corsa ($\epsilon = 0.5$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	0F1, 1E1, 1F1, 361, 3F1, 440, 460	1764	1764	100%	0%	0%	100%	0.178	0.082	6.734
ECU 2	0C1, 0C5, 1C9, 1E9, 265, 2F9, 363, 530	2463	2463	100%	0%	0%	100%	0.186	0.090	5.228
ECU 3	1E5	1538	1538	100%	0%	0%	100%	0.158	0.067	5.398
ECU 4	0C9, 1BC, 1BD, 1C1, 1F5, 2C5, 3D1, 3E9, 3F9, 4C1, 4D1, 772	3366	3366	100%	0%	0%	100%	0.182	0.080	5.228

Table 14
Results with HDBSCAN for Ford Ecosport ($\epsilon = 0.5$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	041, 084, 242, 331, 38D, 3B1, 3B3, 3B4, 3B5, 3B6, 3B7, 3B8, 3C3, 3C7, 3E3, 3EB, 40A, 42C, 447, 581	1316	1316	100%	0%	0%	100%	0.154	0.068	4.323
ECU 2	3A8, 3AA, 3AB, 3AE	1028	1028	100%	0%	0%	100%	0.118	0.044	4.323
ECU 3	3E2, 3EA, 455	64	64	100%	0%	0%	100%	0.165	0.110	2.836
ECU 4	047, 156, 165, 166, 167, 178, 179, 17C, 200, 202, 204, 230, 41F, 421, 424, 42D, 42F, 43D, 43E, 595	5794	5794	100%	0%	0%	100%	0.195	0.135	1.480
ECU 5	049, 04A, 04B, 04C, 076, 077, 07D, 07E, 07F, 082, 083, 091, 092, 213, 214, 216, 217, 23A, 2F1, 326, 332, 333, 35E, 386, 3C8, 3DA, 3E0, 415, 416, 430, 434, 435, 437, 438, 439, 43A, 4B0, 59E	11842	11842	100%	0%	0%	100%	0.142	0.065	1.480

Table 15
Results with HDBSCAN for Dacia Logan ($\epsilon = 0.15$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	1B0, 2BC, 350, 45C, 4AC, 500, 552, 55D, 575, 5DE, 5DF, 657, 69F	3697	3697	100%	0%	0%	100%	0.138	0.058	3.957
ECU 2	186, 18A, 1F6, 217, 2A9, 2C6, 41A, 41D, 511, 5DA, 648, 65C, 66A	6480	6480	100%	0%	0%	100%	0.185	0.076	3.625
ECU 3	090, 0C6, 12E, 242, 29A, 29C, 2B7, 352, 354, 5D7, 666	20187	20187	100%	0%	0%	100%	0.196	0.077	2.732
ECU 4	1A0, 62B	168	168	100%	0%	0%	100%	0.154	0.063	0.789
ECU 5	3B7, 4F8, 646, 6FB	597	597	100%	0%	0%	100%	0.154	0.073	0.789
ECU 6	564, 653	168	168	100%	0%	0%	100%	0.200	0.096	0.885

Table 16
Results with HDBSCAN for Honda Civic ($\epsilon = 0.15$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	039, 305, 401	333	333	100%	0%	0%	100%	0.306	0.159	4.260
ECU 2	1A6, 21E, 221, 294, 295, 309, 372, 374, 377, 378, 386, 405, 428, 42D, 42E	2390	2390	100%	0%	0%	100%	0.208	0.121	0.746
ECU 3	18E	757	757	100%	0%	0%	100%	0.187	0.093	2.812
ECU 4	091, 19B, 1A4, 1AA, 1B0, 1D0, 1EA, 255, 3D9, 406	4218	4218	100%	0%	0%	100%	0.147	0.061	0.746
ECU 5	13C, 158, 17C, 1DC, 1ED, 320, 324, 328, 3D7, 400, 40C, 454, 465	5205	5205	100%	0%	0%	100%	0.183	0.096	3.523
ECU 6	156	1664	1664	100%	0%	0%	100%	0.327	0.180	3.523

Table 17
Results with HDBSCAN for Hyundai i20 ($\epsilon = 0.5$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	043, 044, 383, 593	1164	1164	100%	0%	0%	100%	0.212	0.099	1.176
ECU 2	251, 2B0, 381	3483	3483	100%	0%	0%	100%	0.250	0.092	1.176
ECU 3	18F, 1BF, 200, 260, 316, 329, 492, 4E5, 4E6, 4E7, 545, 547, 549, 556, 557, 5CE, 5CF	7656	7656	100%	0%	0%	100%	0.254	0.104	1.723
ECU 4	153, 164, 220, 386, 387, 507	4888	4888	100%	0%	0%	100%	0.153	0.068	1.723
ECU 5	500, 5A0, 5A1	91	91	100%	0%	0%	100%	0.210	0.095	2.618
ECU 6	4F1, 50C, 50E, 52A, 541, 553, 5B0	485	485	100%	0%	0%	100%	0.249	0.093	2.798

Table 18
Results with HDBSCAN for Hyundai ix35 ($\epsilon = 0.35$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	350	1461	1461	100%	0%	0%	100%	0.188	0.070	6.281
ECU 2	165, 2B0, 5E4	2342	2342	100%	0%	0%	100%	0.289	0.122	4.504
ECU 3	4F0, 690	280	280	100%	0%	0%	100%	0.161	0.068	0.423
ECU 4	153, 164, 1F1, 220, 430, 4B1, 4D0	6210	6205	99.92%	0.08%	0%	100%	0.323	0.157	0.423
ECU 5	0A0, 0A1, 18F, 260, 2A0, 316, 329, 545	9334	9334	100%	0%	0%	100%	0.257	0.142	1.306
ECU 6	429, 5A0, 5A1, 5A2	229	229	100%	0%	0%	100%	0.202	0.085	1.153
ECU 4*	153, 164	-	5	0%	0%	100%	N/A	0.135	0.050	1.137

*indicates samples misidentified by HDBSCAN as a distinct ECU

Table 19
Results with HDBSCAN for Ford Fiesta ($\epsilon = 0.15$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	023, 04A, 04B, 460	498	498	100%	0%	0%	100%	0.209	0.112	1.175
ECU 2	073, 090, 20E, 20F, 211, 212, 213, 215, 216, 2C3, 4B0	8759	8759	100%	0%	0%	100%	0.242	0.124	0.742
ECU 3	150	208	208	100%	0%	0%	100%	0.187	0.057	0.742
ECU 4	190, 275, 2C1, 400, 405, 430, 432, 433, 4E3, 4F2	1398	1398	100%	0%	0%	100%	0.237	0.117	1.402
ECU 5	0FD, 200, 201, 203, 205, 228, 231, 232, 261, 268, 280, 2BA, 360, 364, 420, 424, 428, 4F1	9458	9458	100%	0%	0%	100%	0.189	0.094	1.544
ECU 6	080, 240	1340	1340	100%	0%	0%	100%	0.178	0.075	5.159
ECU 7	455	68	68	100%	0%	0%	100%	0.212	0.085	1.402

Table 20
Results with HDBSCAN for Ford Kuga ($\epsilon = 0.15$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$
ECU 1	0B0, 140	512	512	100%	0%	0%	100%	0.134	0.054	5.282
ECU 2	3E2, 3EA, 455	97	97	100%	0%	0%	100%	0.151	0.062	0.939
ECU 3	050, 06A, 0D0, 0E0, 0F0, 0F5, 100, 2B0	6622	6622	100%	0%	0%	100%	0.196	0.085	1.075
ECU 4	030, 0C8, 150, 17E, 260, 290, 310, 360, 380, 3B4, 400, 405, 40A, 420, 435, 581	3344	3344	100%	0%	0%	100%	0.172	0.075	0.754
ECU 5	060, 070, 080, 090, 0A0, 0C0, 0F8, 120, 130, 138, 1A0, 1B0, 200, 270, 280, 2D8, 2F0, 340	8077	8077	100%	0%	0%	100%	0.170	0.074	0.754
ECU 6	160, 180, 190, 1C0, 1D0, 1E0, 210, 213, 218, 252, 2D0, 2D4	5876	5876	100%	0%	0%	100%	0.229	0.120	0.977
ECU 7	04A, 04B, 208, 388	773	773	100%	0%	0%	100%	0.132	0.054	0.557
ECU 8	2E0	90	90	100%	0%	0%	100%	0.187	0.079	0.512
ECU 9	170, 229, 2A0, 2A5	924	924	100%	0%	0%	100%	0.146	0.067	0.555
ECU 10	010	1007	1007	100%	0%	0%	100%	0.152	0.068	0.566
ECU 11	269	702	702	100%	0%	0%	100%	0.138	0.060	0.512

Table 21
Results with HDBSCAN for John Deere ($\epsilon = 0.15$)

ECU	IDs	Smp.	Cls.	TAR_C	FRR_C	FAR_C	PRE_C	μ_c	σ_c	$d^c_{\min}$	
ECU 1	04EF0021, 0C010321, 0CEF0321, 0CFDCC21, 0CFE4321, 0CFE4421, 0CFE4521, 0CFFFF21, 18EF0021, 18EFFF21, 18F00621, 18FEAE21, 18FEF021, 18FEF121, 18FEF721, 18FEFC21, 18FF9721, 18FFBF21, 18FFC921, 18FFF821, 18FFFA21, 18FFFB21, 18FFFF21, 1CFDDF21, 1CFFFF21	2594	2547	99.92%	0.08%	0%	100%	0.469	0.205	0.372 0.879	*
ECU 2	0CF00300, 0CF00400, 18EF2100, 18EFFA00, 18FEDF00, 18FEEE00, 18FEEF00, 18FEF200, 18FEF600, 18FEF700, 1CEBFF00, 1CECFF00	1231	1127	91.55%	8.45%	0%	100%	0.444	0.214	0.427 0.879	*
ECU 3	18F00503, 1CFEC303	196	196	100%	0%	0%	100%	0.560	0.250	0.688 1.307	*
ECU 4 – ECU24	0CF00400, 18FFBF21, 18FFFA21, 18EF2100, 18FEF600, 18FEDF00, 18FEEF00, 0CF00300, 18EFFA00, 18FEEE00, 18FFF821, 0C010321, 0CFE4521, 18FFC921, 18F00621, 18FFFF21, 18FEF200	-	52	0%	0%	100%	N/A	$\leq$ 0.160	$\leq$ 0.067	$\leq$ 0.441	
Outliers	04EF0021, 0C010321, 0CEF0321, 0CFE4421, 0CFE4521, 18EF0021, 18FEF121, 18FF9721, 18FFBF21, 18FFF821, 18FFFA21, 18FFFB21, 18FFFF21, 0CF00300, 0CF00400, 18EF2100, 18EFFA00, 18FEEE00, 18FEEF00, 18FEF200, 18FEF600, 18FEF700, 1CEBFF00	-	99	0%	0%	100%	N/A	N/A	N/A	N/A	

* indicates the distance to the nearest outlier cluster, while the second value in the cell represents the distance to the nearest main cluster