CANMatch: A Fully Automated Tool for CAN Bus Reverse Engineering based on Frame Matching

Alessio Buscemi, *Student Member, IEEE,* Ion Turcanu, *Member, IEEE,* German Castignani, Romain Crunelle, and Thomas Engel, *Member, IEEE*

***Abstract*—Controller Area Network (CAN) is the most frequently used in-vehicle communication system in the automotive industry today. The communication inside the CAN bus is typically encoded using proprietary formats in order to prevent easy access to the information exchanged on the bus. However, it is still possible to decode this information through reverse engineering, performed either manually or via automated tools. Existing automated CAN bus reverse engineering methods are still time-consuming and require some manual effort, i.e., to inject diagnostic messages in order to trigger specific responses. In this paper, we propose CANMatch – a fully automated CAN bus reverse engineering framework that does not require any manual effort and significantly decreases the execution time by exploiting the reuse of CAN frames across different vehicle models. We evaluate the proposed solution on a dataset of CAN logs, or *traces*, related to 479 vehicles from 29 different automotive manufacturers, demonstrating its improved performance with respect to the state of the art.**

***Index Terms*—CAN Bus, Automated Reverse Engineering, In-Car Networking, Tokenization**

I. Introduction

The Controller Area Network (CAN) is a message-based protocol for in-vehicle communication, released in 1987 to let Electronic Control Units (ECUs) inside vehicles communicate with each other without the supervision of a central bus master. The popularity of CAN has increased to the point that, nowadays, it is considered the de facto standard for in-vehicle communication. Its success mostly derives from its versatility and physical properties, such as robustness to electromagnetic noise. While physical access to the CAN bus within a vehicle is usually possible, the interpretation of its data is rather difficult. The communication is not encrypted and no authentication is implemented for the ECUs. However, the data is encoded according to different formats, which depend on the specific design choices operated by the Original Equipment Manufacturer (OEM) and are kept secret to the general public.

In recent years, the progressive digitalization of vehicles and the release of new technologies in the automotive market – spanning from safety to infotainment – led to a dramatic increase of the number of ECUs connected to the CAN bus. With autonomous driving on the rise and vehicles becoming more and more interconnected to each other and to the infrastructure, we can expect the number of ECUs to further augment in the near future. This will also determine a growth in the quantity of data transiting on the CAN bus. This data already represents a valuable source of information regarding the vehicle, which can be exploited for a multitude of purposes by companies that offer aftermarket solutions: driver profiling, fleet management, and cloud services [1]–[3]. Bertoncello et al. [4] predict that the global market of automotive data, of which CAN is a major source, may be valued between 450 and 750 billion $ by 2030. According to their report, consumers are also interested in automotive data and see it as an enabler of safety features and sustainable development.

A number of solutions have been proposed to address the thirst for clear CAN data, such as SAE J1939 [5] – a set of standards built on top of CAN physical layer that guarantees a standardized communication between ECUs. The popularity of SAE J1939 has increased to the point that, nowadays, a large number of heavy duty diesel vehicles are equipped with it. However, given the unwillingness of the OEMs to disclose the formats of most of the CAN signals of commercial vehicles (cars in particular) to the general public, the most common way to obtain such information is through *reverse engineering*.

Heretofore, this process has been achieved mostly manually [6]. While having been proven to provide reliable results, the manual approach requires from hours to days of intense human work and constant physical access to the vehicle. In addition, it can be intrusive, as one of the adopted techniques is to actively request diagnostic messages to the bus. Recently, researchers have started investigating the automation of this process to make it faster, scalable and standardized [7]–[15].

Aside from the evident advantages that it would bring to the industry, the automation of CAN bus reverse engineering has also gained interest in the scientific community, where automotive cybersecurity is a prominent topic [16]–[18]. A number of wired and wireless attacks based on the injection of malevolent messages in the CAN bus have raised concern in recent years [19]–[21]. In this context, the automation of the CAN bus reverse engineering process plays an essential role. On the one hand, it raises questions about the ease with which adversaries can conduct attacks [22]. On the other hand, it offers a fast and standardized solution that allows researchers who work on intrusion detection to get access to clear in-vehicle data [23]–[26].

In this paper, we present CANMatch, a framework for

Manuscript received XXX, XX, 2021; revised XXX, XX, 2021.

A. Buscemi, G. Castignani, and T. Engel are with the Faculty of Science, Technology and Medicine (FSTM), University of Luxembourg (e-mail: alessio.buscemi@uni.lu; german.castignani@ext.uni.lu; thomas.engel@uni.lu).

I. Turcanu is with the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, and with the Luxembourg Institute of Science and Technology (e-mail: ion.turcanu@list.lu).

R. Crunelle is with Xee/Eliocity SAS (e-mail: rcrunelle@xee.com).

automated CAN bus reverse engineering that is able to decode the format of an unknown vehicle by exploiting frame ID re-utilization. Differently from other proposed solutions – which require injection of diagnostic messages, use of external IMU/GPS data, expertise and partial manual effort – CANMatch gets in input only the CAN raw trace of the vehicle to reverse engineer and clear information on the formats adopted in other vehicle models. CANMatch aims to further speed up and scale the access to clear CAN data for all those researchers and aftermarket companies that drive the innovation in the automotive sector through data-enabled solutions. Our contribution can be summarized as follows:

- We propose CANMatch, a framework that exploits the reuse of frames across different vehicle models to achieve a fully automated CAN bus reverse engineering;
- We present an improved version of the Reverse Engineering of Automotive Data frames (READ) [9] algorithm for tokenization, which increases the tokenization performance by considering the signal endianness;
- We present a clustering-based method that identifies the redundant signals – i.e., carrying the same vehicle functions (although, possibly encoded in a different format) – by finding correlations between previously decoded signals and undecoded tokens. The method also speeds up sensitively the format decoding process.

The paper is organized as follows. Section II provides an overview of the CAN bus reverse engineering process in general and describes the relevant existing related work. Section III presents CANMatch – our proposed solution. Section IV provides a detailed performance evaluation of CANMatch, as well as comparison with the state of the art. Section V discusses the limitations and security implications of our proposed solution. Finally, the conclusions are drawn in Section VI.

II. Preliminary Concepts

We first introduce the general principles of CAN bus communication, describing the message structure and main features that characterize a vehicle function. Then, we summarize the main approaches of CAN bus reverse engineering and the format used to store the decoded information. Finally, we present a detailed analysis of the main related works.

A. CAN Bus

The communication on the CAN bus relies on messages or *frames*. Each ECU periodically sends a CAN frame that, in absence of collisions, is received by all other ECUs. The frames do not contain any information regarding the sending nor the receiving ECU.

The following fields constitute a standard CAN frame (see Figure 1): start of frame (1 bit), identifier (ID) (11 bit in the standard version and 29 bit in the extended one), remote transmission request (1 bit), reserved (2 bit), Data Length Code (DLC) (4 bit), data field (0–64 bit), Cyclic Redundancy Check (CRC) (16 bit), acknowledge (2 bit), and end of frame (7 bit). To be noted that the International Organization for Standardization (ISO) defines an extended CAN frame format as well [27].

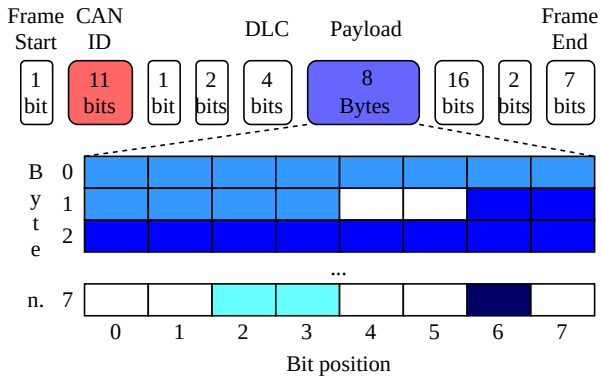

Figure 1. Example of CAN frame with 8-bytes payload. Each color in the payload represents a different signal.

The two most relevant fields for this work are the identifier (ID) and payload. The CAN ID uniquely identifies a frame and carries its priority, which is needed in case of collisions. A collision occurs when multiple messages are simultaneously sent on the CAN bus. Collisions in CAN bus are solved with the use of dominant bits (0) which overwrite the recessive bits(1), always allowing the frame with the highest priority (lowest value) to be received by the ECUs. An ECU can send or receive CAN frames with different IDs, but frames with the same ID are sent by the same ECU.

The payload contains the actual data content of the frame. In CAN standard version, the payload of messages sent with the same ID has always the same length. A payload can contain one or more signals. A *signal* corresponds to a chunk of data that encloses a vehicle function and is defined by the following characteristics:

- **Length** – between one bit for status signals carrying binary information (e.g. the seat belt is fasten or not) and many bits for signals carrying more complex information.
- **Start bit** – at which position/bit within the frame the signal starts.
- **Semantic meaning** - what is the actual vehicle function encapsulated in the frame. A signal can express an event-driven physical phenomenon (e.g. vehicle speed) or an internal function of the vehicle (e.g. cyclic counters).
- **Endianness** – is the order of bytes in which the cross-byte signals (signals that span across multiple consecutive bytes) can be represented. In the Big Endian (BE) format, the most significant byte appears first in the frame, while in the Little Endian (LE) it appears last [28].
- **Signedness** – a signal is signed if the numerical values it carries are both positive and negative, and unsigned if only positive. Signed signals are typically represented in two's complement encoding.
- **Scale factor and Offset** – typically, to find the actual human-interpretable physical value v carried by a signal s, it is not sufficient to parse the signal encoding from binary (or hexadecimal) to decimal format. Letting r be the raw decimal value of the signal, to obtain v we need to apply a scale factor f and add an offset o, as shown

in Equation (1)

$$v_s = f_s \cdot r_s + o_s \tag{1}$$

To be noted that factor and offset of a signal can be equal to, respectively, 1 and 0. In such case, the signal is immediately interpretable.

In CAN standard version, all these properties are fixed and do not change over time. In other words, once they are known for a certain signal, we can use them to correctly identify and interpret the content of the signal at any time. According to [9], [13], signals can be grouped into the following categories, based on their function:

- **Physical** - signals that corresponds to telemetries or other sort of physical measurements, such as vehicle speed or engine coolant temperature.
- **Status/multi-value** - typically signals between 1–4 bit long. They either express a binary property or a limited set of options (e.g. the windscreen wiper speed).
- **Counters** - some signals behave as cyclic counters.
- **Checkcodes** - in addition to the CRC field, some frames contain supplementary checkcodes within their payload.
- **Constant/unused** - some payloads present chunks of consecutive bits that do not change over time. They might correspond to signals triggered only under exceptional conditions not observable during a standard data collection session (e.g. the activation of the airbag), or simply be unused bits.

B. CAN Bus Reverse Engineering

The aim of reverse engineering is to identify the location of the signals within the frames, understand their semantic meaning, i.e., what is the vehicle function that they represent, and decode their format. This can be achieved by performing a certain number of *manual* operations, such as activating/deactivating vehicle sensors to trigger events in the CAN bus. The changes in the CAN traffic caused by these events can be spotted by a human operator by using a number of tools (e.g. Wireshark) in real time or through an a posteriori analysis [29], [30]. Reverse engineering can also be performed by injecting Parameters IDs (PIDs) via the On-Board Diagnostics (OBD-II), which is a law-enforced port present in all vehicles mostly used for retrieval of emission-related parameters and diagnosis purposes. Note that in absence of other access points to the bus, this interface can also be employed to log other CAN data.

The *automated* reverse engineering approach is mostly based on correlating in-vehicle data with some live clear data provided by GPS/Inertial Measurement Unit (IMU) sensors and/or the automated injection of PIDs. Some algorithms, in particular the ones based on Machine Learning (ML), exploit CAN data collected on vehicles previously reverse engineered.

Usually, automated reverse engineering tools perform three main tasks: *tokenization*, *translation* and *format decoding*. The goal of tokenization is to identify the boundaries of the signals within the payload of each CAN frame. Most tokenization algorithms also provide the type of signals, as described in Section II-A. A *token* is a signal of which we know the boundaries and endianness, but whose format and vehicle function have not been decoded yet. The goal of translation, instead, is to identify the vehicle function carried by tokens – which can also be referred to as *signals*. Finally, the format decoding procedure discloses the scale factor, offset, and unit (for physical signals), which ultimately allows identifying the information carried by the signal. In this work, we focus on all three phases.

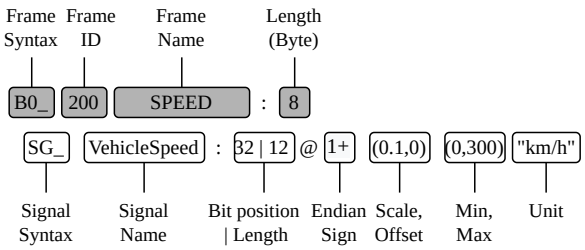

Figure 2. An example of format description in a DBC file.

To evaluate a CAN bus reverse engineering method on a target vehicle, apart from its CAN trace, it is necessary to have the ground truth to validate the results against. The most widely employed standard to store such information is Database CAN (DBC), released by Vector Informatik [31]. A DBC file stores the characteristics described in Section II-A, but also the maximum and minimum values that a signal can assume. Figure 2 illustrates an example of format description in a DBC file. The top part of the figure, highlighted in grey, shows the syntax of a frame, while the part below represents an example of a signal within the frame.

Typically, the *original* DBC files owned by OEMs are not accessible to the general public. While validating a reverse engineering tool on original DBC files would certainly help validating the developed solutions, unfortunately, obtaining such information for research purposes requires non-negligible efforts and long negotiations with the OEMs. Moreover, to make sure the proposed solutions are universal, one would have to obtain multiple DBC files from different OEMs. An alternative option for obtaining ground truth datasets for validation is to employ DBC files obtained by third-party expert technicians through manual reverse engineering. We define these DBC files as *generated*. The disadvantage of generated DBC files is that, generally, they do not cover all the information related to the target vehicle, as some signals may have not been spotted or fully decoded. In this work, we validate our solution using a generated set of DBC files, described in Section IV-A.

C. Related Work

Jaynes et al. [7] are the first to tackle the automation of the CAN bus reverse engineering process. The authors propose training an ML classifier to identify the sender ECU based on features extracted from the payloads of the frames. In their work, the payloads are analyzed as a whole, instead of identifying and interpreting the signals contained in them, thus making this approach not universally applicable (as ECUs from different manufacturers encapsulate different assortments of signals in the frame payloads).

In concurrent works, Nolan et al. [10] and Marchetti and Stabili [9] investigate tokenization through bit flip counts. A bit flip corresponds to a change in the value of the bit (from 0 to 1 or vice versa). The assumption behind this approach is that Least Significant Bits (LSBs) flip more often than Most Significant Bits (MSBs) due to the own nature of the signals. In particular, Marchetti and Stabili [9] calculate the *magnitude* of a bit b as $\log_{10}(BFR_b)$, where Bit Flip Rate (BFR) corresponds to the bit flip count divided by the frame time series length. The list of the magnitude extracted for all bits in a frame is scanned from start to end. When the magnitude of a bit is higher than the magnitude of the following bit, a boundary is found. Nolan et al. [10] consider only physical signals, while Marchetti and Stabili [9] consider also constant, multi-values, counter and checkcode signals. In particular, counters are signals whose BFR keeps doubling from the most to the least significant bits, while in checkcodes the BFR of all bits is normally distributed around 0.5. Nolan et al. [10] make a first attempt to decode the endianness of the signals. However, in the identification of the little endianness, the authors wrongly consider a reverse order of the bits instead of the bytes which, instead, truly characterizes the endianness.

Huybrechts et al. [11] present two different approaches to perform both identification of the signals boundaries and translation of the signals. In the first approach, they are the first to exploit OBD-II PIDs to identify signals. The second approach makes use of GPS data to correlate with CAN data, employing a Long Short-Term Memory (LSTM) network for the translation task. However, in this work, the authors incorrectly assume that signals can be only 1 or 2 Byte long and that their boundaries coincide with, respectively, the LSB and MSB of the bytes in which they are contained.

Verma et al. [12] introduce Automatic CAN Tokenization and Translation (ACTT), which simultaneously tokenize, translate and decode the format of the signals, by matching CAN data with live diagnostic data obtained through the injection of OBD-II PIDs. In the preliminary phase, ACTT adopts a brute-force approach: it initially considers all possible combinations of start and end bits of (non-constant) signals within a frame. Subsequently, a linear regressor attributes a fitness score to each of these signals. Finally, a dynamic programming algorithm extracts the set of non-overlapping signals which maximizes the total fitness score.

Pesé et al. [13] propose LibreCAN, a tool for complete CAN bus reverse engineering, composed of three phases. The first phase performs the tokenization task through a modified version of the READ algorithm [9]. The second phases concerns the translation of powertrain signals (e.g. engine RPM). The translation task is performed through a cross-correlation with data collected via OBD-II PIDs and IMU external sensors installed in the vehicle. The scale factor and offset of signals is found through linear regression. The third phase introduces a semi-automated procedure for body-related signals (e.g. door status, headlights status etc.). A human operator iteratively activates sensors in the vehicle to trigger changes in the signals related to them, with the purpose to spot differences against a benchmark trace (i.e. in which no action was performed). At the time of writing, LibreCAN is considered the most complete tool for CAN bus reverse engineering. It has been tested on real CAN data collected on four cars and validated against the original DBC file from the manufacturer itself, obtaining convincing results. Nonetheless, it is not clear whether this method would perform as well on cars from other makers. Unlike LibreCAN, which requires the same (significant) amount of manual work to collect data for each new CAN trace to interpret, CANMatch exploits the DBC files of reverse engineered vehicles to iteratively speed up both data acquisition and decoding phases.

In our previous work [14], we propose a first attempt towards the translation of physical signals by using CAN data only, i.e., without the injection of OBD-II PID nor the collection of IMU/GPS data to match with the CAN trace. The underlying assumption is that signals related to the same vehicle function present characteristics that are intrinsic and caused by the physical phenomenon they represent. These properties can be used to recognize the semantic meaning of the signals, despite their different format (i.e. scale factor and offset) and environmental conditions (driving style, traffic etc.) in which the data are collected. These environment-agnostic features are used to train an ML classifier to recognize the semantic meaning of critical signals. The main limitation of this work is that it focuses solely on the translation task, thus already assuming perfectly extracted tokens.

Ezeobi et al. [15] introduce clustering in the scope of CAN bus reverse engineering. In this work, several clustering techniques, among which DBSCAN [32] and Agglomerative Hierarchical Clustering (AHC) [33], are adopted to identify sets of frames related to the same vehicle function.

In 2021, Choi et al. [34] challenge some of the assumptions on which READ [9] is based, such as the distribution of BFR in the checksum signals and the monotonic increase of BFR from the MSB to the LSB of physical signals. The authors evaluate their approach on a set of four vehicles from the same OEM and validate it against OpenDBC, an open-access repository of generated DBC files [35]. On the considered dataset, this method outputs equal results on counter signals and performs better on physical and checksum signals. However, in this work the authors do not address the endianness of the signals.

All the existing solutions that achieve complete CAN bus reverse engineering (tokenization, translation and format decoding) require manual effort, installation of external sensors (GPS/IMU) and injection of diagnostic messages through the OBD-II port at data collection time. These characteristics make these solutions intrusive and only partially scalable, as the amount of actions to be performed by a human operator and the time necessary for the acquisition of data is constant for every new vehicle to reverse engineer. In this paper, we propose a non-intrusive tool, whose performance and time required for data acquisition and execution improve with the number of vehicles that are reverse engineered. Our approach is based on the reuse of information obtained on these vehicles (in the form of a DBC file), and guarantees high scalability compared to other proposed solutions.

III. CANMATCH

It is common knowledge that different car manufacturers equip their vehicles with ECUs from a limited number of Tier-1 suppliers. It follows that an ECU can be part of the electronic system of multiple vehicle models. Intuitively, this is mostly true for models from the same brand or alliance, to enhance economies of scale. But, it can also be valid for models of different OEMs that get their supplies from the same Tier-1 supplier.

What does not seem to be public knowledge is that the CAN frames associated with these ECUs are also shared by different vehicle models. The direct implication of the reuse of CAN frames is that, by correctly reverse engineering signals within a frame with a certain ID in a vehicle model, we can assume that frames with the same ID found in other vehicles carry the same signals.

CANMatch exploits the high reuse of frame IDs across different vehicle brands to reverse engineer the signals contained in their payloads. Provided that there is a ground truth dataset containing already decoded CAN traces (i.e., by using any of the methods described in Section II-B), CANMatch decodes an unknown CAN trace by performing the following three main phases (illustrated in Figure 3):

- **Phase 1 (Frame Matching)** – CAN IDs and payloads are extracted, along with their timestamp. CANMatch searches for a match between the IDs found in the trace (to be decoded) and the same IDs present in the ground truth (previously decoded signals). If a match occurs, the frame is decoded according to the signals extracted from the matched frame in the ground truth dataset.
- **Phase 2 (Tokenization)** – While part of the signals present in the trace have been decoded through Phase 1, portions of the trace are still not decoded (i.e. we do not know which signals they contain). The tokenization task is performed on such unknown areas of the trace. The output is a set of tokens, represented by their boundaries and their endianness. To be noted that the signal each token is carrying is still unknown at the end of Phase 2.
- **Phase 3 (Redundancy Resolution)** – Redundant signals are signals that carry the same vehicle function/telemetry. CANMatch looks for correlations between the signals decoded via Phase 1 and the unknown tokens. If a match occurs, the vehicle function/telemetry of the known vehicle is attributed to the token. The scale factor and offset of the token are decoded too.

A. Phase 1

The goal of Phase 1 is to decode signals by matching the frames of the vehicle to reverse engineer with frames having the same ID, and extracting their content. The first step is to parse the CAN raw data into a standardized format and extract all the frame IDs. The second step is to query the ground truth dataset to find the same CAN IDs on previously reverse engineered vehicles. If all the frames associated with the same ID found in the ground truth have the same content, the matching is straightforward: the signals and all information related to them are assigned to the frame. This information includes the semantic meaning of the signals, their position within the frame, and their format (i.e. endianness, offset, scale factor and unit). In other words, the frame is decoded. An analysis of our ground truth set has revealed that 53.9 % of the frames associated to the same ID contain the same set of signals.

Vice versa, when different content is associated with the same frame ID for different vehicles in the ground truth, CANMatch needs to deals with the ambiguity. It does so by taking into account different properties of the frames. Specifically, all frames with a different length (as indicated by the DLC field) than the one currently analyzed are preliminarily discarded. Intuitively, two frames with different lengths cannot contain the same set of signals. Then, a likelihood score is calculated taking into consideration the number of frame IDs in common between the target vehicle and the vehicles containing that same ID. The reasoning behind this approach is that vehicles with many common IDs have higher probability to be equipped with the same ECUs and, therefore, to share the same signals (see Section IV-A). This is especially true for vehicle models produced by the same OEM. Finally, the frame associated with the highest likelihood score is considered as the reference frame. Similarly to direct frame matching, all the signals and their relative format found in the ground truth of the reference frame are attributed to the CAN ID of the target vehicle.

B. Phase 2

The trace chunks that are not decoded by Phase 1 (i.e., no matching frames are found) are processed in Phase 2. This phase performs the tokenization task, whose goal is to identify the boundaries and endianness of tokens within an unknown frame. The tokenization process is based on the same assumption made in [9]: differences among two consecutive values of a signal representing a physical phenomenon are small, due to the own nature of the phenomenon. Following this principle, the least significant bit, which impacts the least on the value of the signal, usually flips (changes its value) more often than a more significant bit of the same signal.

The pseudocode summarizing the implementation of Phase 2 is illustrated in Algorithm 1. The algorithm extracts all the

Algorithm 1 Phase 2

Input: Undecoded Trace Chunks U
Output: A set of tokens T

```
T ← []
for id in get_frame_IDs(U) do
    ts ← get_frame_time_series(U, id)
    bfr ← calculate_bit_flip_rate(ts, U)
    be ← tokenize_big_endian(bfr)
    le ← tokenize_little_endian(bfr)
    be_types, le_types ← get_tokens_type(be, le)
    if score(be) ≥ score(le) then
        T.append(⟨be, be_types, 0⟩)
    else
        T.append(⟨le, le_types,1⟩)
    end if
end for
```

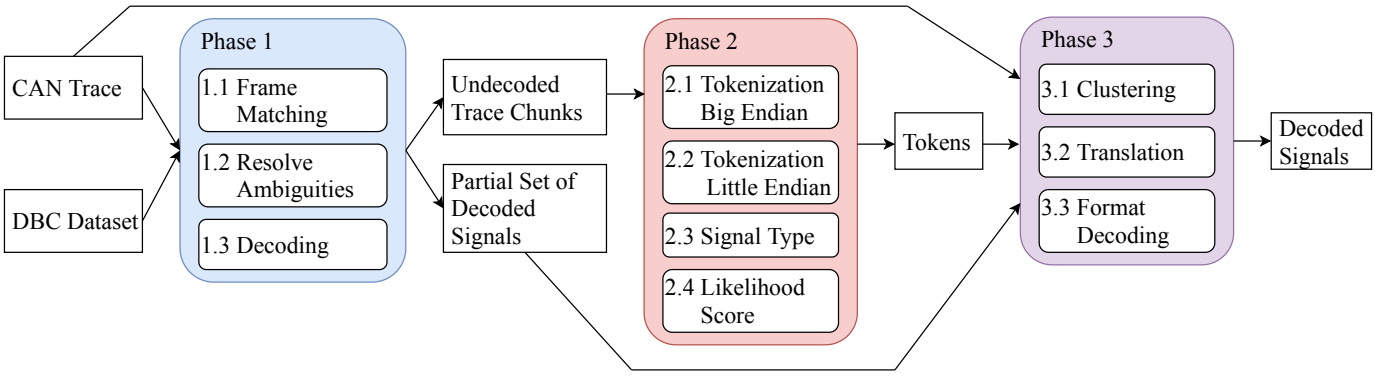

Figure 3. CANMatch pipeline: the framework takes in input a raw CAN trace and provides in output a list of decoded signals.

frame IDs from the trace and splits the trace into n sub-traces, where n is the number of unique IDs in the trace (line 3). Each of these sub-traces contains the payload of frames associated to the same ID, in the same temporal order as they appear in the trace. Then, the sub-traces are iteratively processed independently of one another. For each sub-trace, the BFR of each bit in the frame is extracted (line 4). The BFR corresponds to the total number of bit flips in consecutive payloads divided by the number of payloads in a considered time span.

Once all BFRs within a frame are computed and stored in an array according to their position in the frame, the algorithm proceeds to extracting the boundaries and endianness of the tokens. Regarding the endianness, we make the following assumptions, based on the results of the analysis presented in Section IV-A:

- A CAN trace can contain signals encoded with both big and little endian formats.
- Signals within a single frame are encoded using the same format.

A set of tokens is extracted assuming one endianess format at a time (lines 5–6). For the BE encoding, the BFR array is scanned from the beginning to the end, seeking for a drop in the BFR between consecutive elements. We consider a drop in the BFR value if the difference between two consecutive elements is higher than a tolerance threshold τ:

$$|a_i - a_{i+1}| > \tau \tag{2}$$

where a_i is the i^{th} element of the array. When such a decrease is found, a boundary between two tokens is identified. The goal of τ is to prevent the premature identification of boundaries between two tokens caused by small statistical fluctuations in the BFR.

For the LE format, the algorithm first reverses the bits in each individual byte, without changing the order of the bytes. Through this approach, we do not try to reconstruct a priori the correct order of the bytes. Instead, we aim to exploit the widely-adopted technique [9], [13] of identifying continuity (or disruption) in the BFR along the whole frame also in the case of a LE format. In fact, as for BE, the array is scanned to find decreases in the BFR values. But, due to the different order of the bytes of the LE format, the scan is performed from the end to the beginning of the BFR array. For the sake of comprehension, it should be noted that the reversing of bits within the bytes is exclusively carried out with the purpose of evaluating the likelihood of the two endianness formats.

Differently from related work [9], [10], which mistakenly consider the endianness as a property related to the order of the bits (instead of the bytes), here the bits are reversed at *Byte-level* and not considering the frame as a whole. As a matter-of-fact, after the tokenization is performed, the BFR array is brought back to its original form and the boundaries are updated accordingly. Figure 4 shows an example of tokenization on a 2-Byte payload performed according to the two endianness encodings.

Subsequently, the algorithm pursues to identify the token types (see Section II-A) in the two sets (line 7). Multi-value, checkcode and constant tokens are labelled in a similar fashion to [9], [13] presented in Section II. By employing simple techniques for time series analysis, the algorithm is able to identify counter tokens according to the following properties: (i) counters that exhibit a seasonal trend, and (ii) counter cycles that are characterized by a monotonic growth followed by a sudden drop (the counter is reset). Tokens that do not fall in any of the previous categories are labelled as physical.

Once the token types are found, the algorithm decides which endianness is the correct one and, subsequently, which is the final set of tokens to extract from the frame. For this purpose, a *likelihood score*, S_{L}, is calculated on the tokens of both outputs. S_{L} depends on two main components (subscores): (i) the number of cross-byte tokens, and (ii) a volatility component that penalizes highly-volatile tokens.

To find the first subscore, the algorithm computes a preliminary score S_{c}, defined as:

$$S_{\text{c}} = |T'| + \alpha \sum_{t \in T'} (S_t + E_t) \tag{3}$$

Here, T' corresponds to the set of cross-byte tokens (i.e., tokens contained in different consecutive bytes). We expect the tokenization algorithm to find a higher number of these signals when the endianness is the correct one. S_t and E_t are binary variables defined as follows:

$$S_t = \begin{cases} 1, & \text{if } start\_bit_t \pmod 8 = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

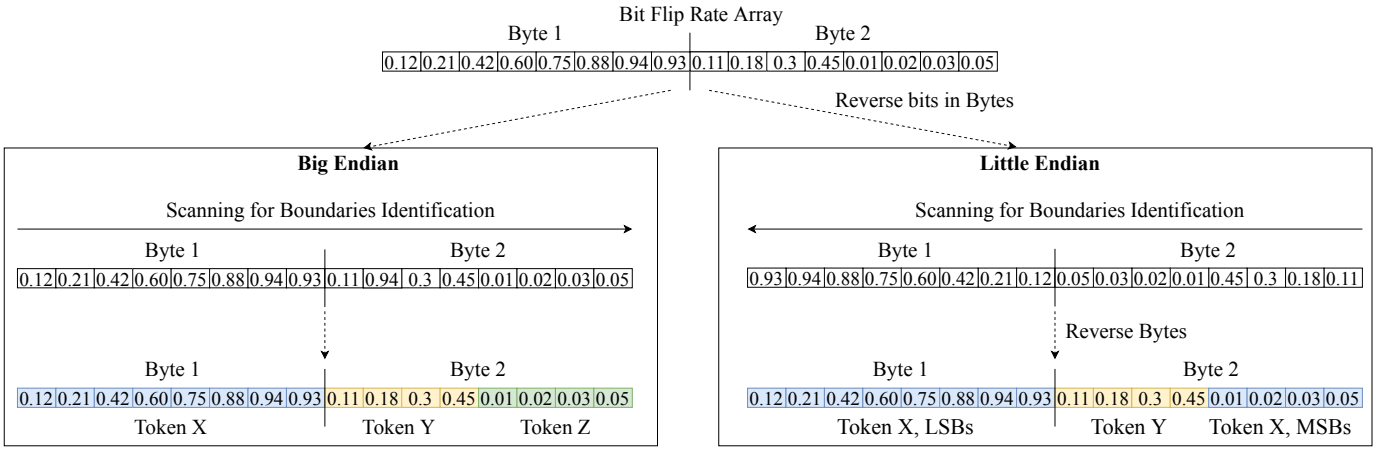

Figure 4. Tokenization according to different types of endianness: (i) assuming big endian (left figure), two distinct one-byte-long tokens are found; (ii) assuming little endian (right figure), one two-byte-long token is found.

$$E_t = \begin{cases} 1, & \text{if } (end\_bit_t - 1) \pmod 8 = 0 \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

To be noted that the second component of Equation (3) is weighted by α in order to incentivize tokens that start and/or finish exactly at the least and/or most significant bit inside the frame. While the CAN protocol allows signals to start and end at any position within a byte, the start and end bits of long signals often coincide with, respectively, the least and most significant bit of a byte. S_c is then normalized between 0 and 1. We call this normalized score S'_c.

The volatility component, S_v, can be computed as follows:

$$S_\text{v} = \frac{1}{|T'|} \sum_{t \in T'} V_t, \quad (6)$$

where V_t corresponds to the volatility of a single token in T'. It is in the range [0,1], where values that tend to 1 indicate a static behaviour, while values that tend to 0 express a highly volatile behaviour. The reason we consider the volatility component is because signals encapsulate information that has a physical meaning. If a token is correctly extracted, i.e., its boundaries and endianness are correct, its time series perfectly describes the behaviour of the vehicle function/telemetry it carries over time. By contrast, cross-byte tokens that are not correctly extracted usually contain bits belonging to different signals. The time series of such tokens do not represent any physical information and, as a consequence, are characterized by a highly volatile and unpredictable behaviour. Our assumption is that, among different sets of tokens computed for the same frame ID, the set that produces tokens whose time series are less volatile on average has a higher chance to be the correct one.

Finally, the likelihood score is computed as follows:

$$S_\text{L} = S'_\text{c} + \beta \times S_\text{v}, \quad (7)$$

where β assigns a weight to the volatility component.

C. Phase 3

Some signals sent within the CAN bus encapsulate the same vehicle function. This is especially true for critical signals such as vehicle speed and throttle pedal position. These signals, carrying the same vehicle function, hereafter referred to as *redundant*, are typically represented with a different format and/or refer to another measurement unit (e.g. $\mathrm{mi/h}$ and $\mathrm{km/h}$ for the speed). The goal of Phase 3 is to identify and decode signals among the tokens extracted by Phase 2 that bring the same vehicle functions as the signals decoded by Phase 1.

To assess whether two or more CAN signals are redundant, it is necessary to calculate a score based on the similarity of their time series, as extracted from the trace, and evaluate it against an acceptance threshold. In our case, the most straightforward approach would be a brute force one, i.e. calculate the score between each token and each signal decoded in Phase 1. Nonetheless, by definition, this approach is computationally inefficient. Phase 3 reduces the time needed to find these correlations by preliminarily clustering the time series of tokens and signals together. Then, the similarity score is only calculated between each token and the reference signal within the same cluster. In principle, just applying the clustering would be enough to identify the tokens that are redundant. However, as explained with more detail in Section IV-D, the clustering involves the risk of missing out the identification of some signals if not perfectly tuned. Once the vehicle function is identified, the token is fully decoded by calculating its format – scale factor and offset – through linear regression.

The pseudocode of Phase 3 is shown in Algorithm 2. Prior to the computation of the similarity scores, all the time series are interpolated in n points (line 3). The interpolation reduces the number of points in the time series and, therefore, contributes to a faster computation of the similarity scores. While the majority of CAN frames are sent with a periodicity between 10–50 ms, the information they carry is related to physical phenomena generated within the vehicle and, therefore, exhibit trends that can be encapsulated with one point per second granularity, which we consider is a good trade off between computational cost and information loss.

Redundant signals usually differ from one another in format. To minimize the impact of the scale in the computation of the similarity score, all time series are also scaled between 0 and 1

(line 4). Once the tokens and decoded signals are interpolated and scaled, the similarity score can be calculated between each of the tokens and each of the decoded signals within the same cluster (lines 5–13).

It may happen that a token is found to be similar to more than one decoded signal. This typically happens if the decoded signals are semantically related or if the threshold value θ is set too high. In the case of such an ambiguity, the token is attributed to the vehicle function of the decoded signal with the best (lowest) score (line 9).

Finally, Phase 3 calculates the scale factor and offset of the translated token (line 10). This is achieved by feeding a linear regression with the time series of the token and the time series of the redundant signal of reference. The slope of the function outputted by the linear regressor corresponds to the scale factor, while the intercept corresponds to the offset. Decoding the format of CAN dynamic signals by using linear regression was firstly introduced in [13]. In their work, the reference signals are the IMU and OBD-II data collected along with the CAN traffic and can be passed to the linear regressor as their values are directly interpretable by humans. In our case, instead, prior to the regression, the raw values of the reference signal have to be parsed to the human-readable format using the factor and offset extracted in Phase 1. As mentioned in Section II-A, signals can have null factor (i.e. equal to 1) and null offset (i.e. equal to 0). According to the analysis on our ground truth set on 477 generated DBC files (see Section IV), 9.7% of the dynamic signals have such a format. For these signals there is no need to conduct the preliminary parsing.

To be noted that signals representing the wheels speed are often mislabelled as vehicle speed and vice versa. This is due to the intrinsic semantic correlation between these telemetries. To address this issue, the algorithm exploits a characteristic that helps distinguishing them apart. In particular, signals related to the wheels speed are often twinned, meaning that they are carried by the same frame, while those representing the vehicle speed are not. Phase 3 post-processes the results of the translation to exploit this peculiarity and, therefore, increase the overall accuracy (line 14–21). Vice versa, in the scope of the format decoding, each signal with a strong correlation to the vehicle speed can be used to decode the others. For instance, the front right wheel speed can be used as reference signal to decode the format of other wheels and vehicle speed.

Algorithm 2 Phase 3

Input: Input Trace I, Partial set of Decoded Signals *PDS*, Tokens T
Output: *DS* (Decoded Signals)
1: *DS* ← *PDS*
2: *T_ts*, *PDS_ts* ← extract_time_series(*I*, *T*, *PDS*)
3: *T_its*, *PDS_its* ← interpolate(*I*, *T_ts*, *PDS_ts*)
4: *T_ists*, *PDS_ists* ← scale(*I*, *T_its*, *PDS_its*)
5: *C* ← clustering(*T_ists*, *PDS_ists*)
6: **for** *c* **in** *C* **do**
7: *CS* ← *c*.get_signals()
8: **for** *t* **in** *c*.get_tokens() **do**
9: *vf*, *mss* ← translation(*T_ists*.get(t), *T_ists*.get(CS))
10: *f*← format_decoding(*T_its*.get(t), *T_its*.get(mss))
11: *DS*.append(<*t*, *vf*, *f*>)
12: **end for**
13: **end for**
14: *S* ← get_speed_signals(*DS*)
15: **for** any s_i, s_j **in** *S* **do**
16: **if** s_i.get_frame_id()==s_j.get_frame_id() **then**
17: *DS*.update_vehicle_function(s_i,'WheelSpeed')
18: **else**
19: *DS*.update_vehicle_function(s_i,'VehicleSpeed')
20: **end if**
21: **end for**

IV. Performance Evaluation

We first provide a preliminary data analysis of the considered dataset and describe the data collection process. Then, we present the performance of each of the three phases, evaluated independently (i.e., the evaluation of each phase does not depend on the evaluation results of the previous phase). Finally, we evaluate the performance of the entire CANMatch framework where all the phases are linked together. All the running times reported in the following sections refer to executions performed with a Dell Latitude 5490 laptop, equipped with Intel(R) Core(TM) i5-8250U CPU @ 1.60 GHz, 1800 MHz, 4 Core(s) with 8 Logical Processor(s).

A. Data Collection

We evaluate the performance of CANMatch based on two ground truth datasets, as follows:

- Set of Derived DBC (SDDBC) – a set of 477 generated DBC files related to vehicle models from 28 different makers from EU, USA, Japan and South Korea.
- OpenDBC, a set of generated DBC files publicly available online [35].

As raw CAN traces to be decoded, we use three sets as follows:

- Parked Vehicles Traces (PVT) – a set of 477 CAN raw traces. These traces are 10 s long and are collected on vehicles being in idle mode, i.e., parked vehicles with no human action being performed. The ground truth of each of these vehicles is contained in SDDBC.
- Driven Vehicles Traces (DVT) – a set of 13 CAN raw traces. These traces are 1-2 min long and are collected on moving vehicles. The ground truth of each of these vehicles is contained in SDDBC.
- OpenDBC DVT (ODVT), a set of 2 CAN raw traces. These traces are 1 min long and are collected on moving vehicles. The ground truth of each of these vehicles is contained in OpenDBC.

SDDBC has been generated and labeled through manual reverse engineering – using the methodology presented in Section II-B – by a partner company operating in the sector of automotive telematics services. To the best of our knowledge, with an aggregated number of 21 526 decoded signals related to 477 vehicles, this is the most extensive and diverse set of generated DBC files commercially available. Differently to SDDBC, not all signals present in OpenDBC files have been

Table I
DVT AND ODVT SET

Set	Vehicle Model	# Signals DBC	Length (s)
DVT	Audi A3 2012-	59	60
	BMW X1 2015-	53	60
	Citroen C3 Picasso 2009-	46	120
	Fiat Qubo 2008-2019	26	120
	Kia Sportage 2016-	31	60
	Mercedes A-Class 2018-	59	60
	Nissan Micra 2005-2011	41	120
	Peugeot 208 2011-	47	120
	Peugeot 307 2005-2008	37	60
	Renault Captur 2013-	46	120
	Renault Megane 4 2016-	48	60
	Smart Fortwo 2014-	57	120
	VW Golf 5 2003-2009	63	60
ODVT	Acura ILX 2013-	177	60
	Volvo V40 2017-	54	60

fully decoded. For some signals, only the location is identified, with no information concerning semantic meaning nor the format. For the sake of the validation, we have preliminarily discarded these signals from the ground truth.

PVT is employed for the evaluation of Phase 1 (see Section IV-B). Unfortunately, PVT cannot be used to also evaluate Phase 2, because all event-driven signals related to a moving vehicle do not display any change over time when the vehicle is parked and, therefore cannot be detected by the tokenization algorithm. Similarly, the time series of these signals do not present distinctive features that can be used in the clustering and translation tasks performed by Phase 3. As a consequence, we use DVT and ODVT for the validation of Phase 2 and Phase 3, as well as the entire pipeline. Information on DVT and ODVT is presented in Table I.

All traces in PVT and DVT are extracted with a `PCAN-USB FD` dongle [36], while traces in ODVT are collected with different dongles. Nowadays, many vehicles are equipped with a gateway between the OBD-II port and the CAN bus, whose purpose is to prevent the logging of non-diagnostic data. To sniff data from vehicles equipped with such a gateway, we have identified and attached the dongle directly to the CAN bus wires. In the rest of the vehicles, we have simply connected the dongle to the OBD-II port.

Finally, all tests presented in the following sections are performed under real-world conditions, but the results refer only to the decoded signals present in the ground truth.

B. Phase 1 Evaluation

We evaluate the performance of Phase 1 by testing each trace in PVT through a *leave-one-out-cross-validation* approach. In other words, each trace is iteratively considered as belonging to an unknown vehicle to reverse engineer, whose ground truth is removed from the DBC dataset.

We analyse two performance metrics, *recall* and *precision*. The recall is defined as the number of correctly-decoded signals divided by the total number of signals present in the ground truth for the considered vehicle. The precision is defined as the number of correctly-decoded signals divided by the total number of decoded signals. These metrics highlight two key

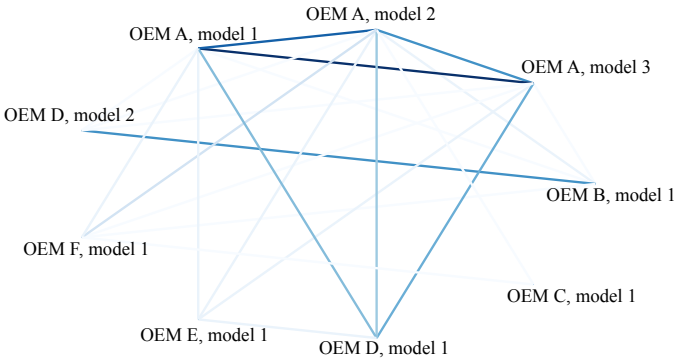

Figure 5. Example of reuse of CAN frames among vehicles from different constructors. The thicker the line, the more CAN IDs are shared between the two models.

aspects to take into consideration while evaluating CAN reverse engineering: how extensive is the set of outputted decoded signals (recall), and how reliable is this output (precision). Other metrics widely employed in literature, such as F-Score, can be derived from recall and precision.

From a preliminary analysis of SDDBC, it seems that a majority of the CAN IDs are reused in different vehicles, and so are the signals carried in their payloads. Specifically, we discovered that, out of 588 total frame IDs in the dataset, only 151 are uniquely found in only one vehicle, while all the others can be found in two or more vehicles.

To visualize effectively the relations between the different vehicles in terms of common frame IDs, we represent our dataset via a relation graph. A reduced version of this graph generated with nine (anonymized) vehicles only is illustrated in Figure 5. The nodes of this graph represent the vehicle models. There is an edge between any two nodes in the graph if the corresponding vehicles share at least one CAN frame ID. The edge thickness represents a weight corresponding to the number of common CAN frame IDs between two vehicle models. The full graph contains a total of 477 nodes and 8683 edges (not shown for readability reasons).

To understand how the size of the ground truth dataset influences the performance of Phase 1, we tested the algorithm on ground truth of different dimensions, starting from a ground truth composed of only one vehicle. The vehicles that constitute each subset are extracted randomly. To obtain results independent from the choice of the particular subset of vehicles, we perform this random selection 10 times for each ground truth size. It follows, that each vehicle in the PVT set is tested once when assuming the total ground truth, and 10 times when considering each of the inferior ground truth sizes.

Figure 6 illustrates the average values and 95 % confidence intervals obtained for all 477 vehicles and all iterations on ground truth size. With a ground truth set of only one vehicle, less than 20 % of the signals are correctly matched on average. The much higher precision (80 % circa) suggests that when a match occurs between a CAN ID in the vehicle to reverse engineer and a vehicle in the ground truth set, we can be fairly confident that the match is correct. As expected, both precision and recall increase with the dataset size. With a ground truth size comprising more than 25 vehicles, we can

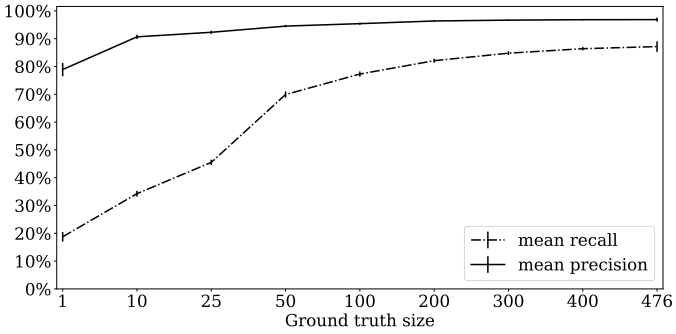

Figure 6. Mean recall and precision scored by CANMatch on datasets of different size.

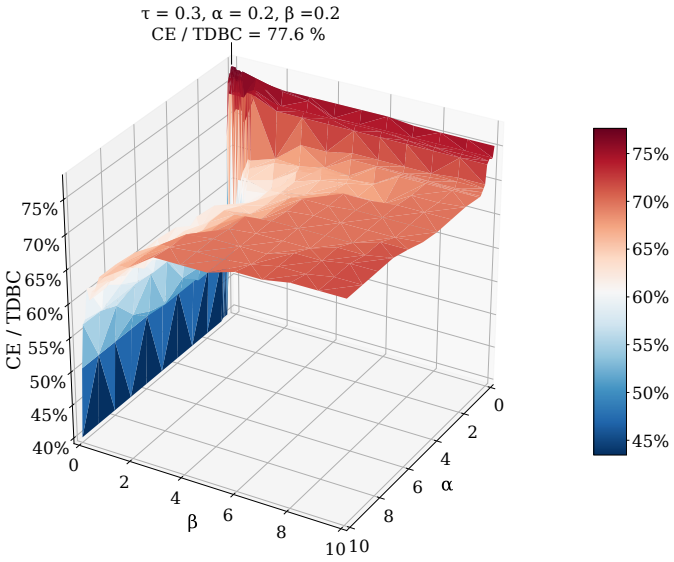

Figure 7. Phase 2 performance evaluation: tuning of weight parameters α, β, and τ.

expect more than half of the signals to be decoded on average, with a precision superior to 90 %. Almost optimal recall and precision are achieved with a ground truth size of 200. After this threshold, the improvement in performance according to both metrics is marginal.

When considering all DBC files in our dataset (minus the one related to the current tested vehicle) as ground truth set, we obtain a mean recall of 83.3 % and a mean precision of 97.7 %. The very high precision shows that almost all ambiguities are correctly solved. The lower recall is mostly due to the presence of unique frame IDs in the dataset. If a tested vehicle has one or more frames with an ID not present in the ground truth set, the signals within that frame cannot be decoded. Finally, it is to be noted that the frame matching provides the exact scale factor and offset for the given signals. It follows that the format decoding results can be directly inferred from the aforementioned results.

C. Phase 2 Evaluation

To evaluate Phase 2 independently from the results obtained with Phase 1, we test it on each trace of DVT and ODVT, without prior processing (as if Phase 1 did not decode any signals). In [13], the authors define three metrics: correctly extracted tokens (CE), total number of signals in the DBC file (TDBC) and total extracted signals (TE). Then, they evaluate their tokenization algorithm according to the ratios CE/TE and TE/TDBC. In the case of LibreCAN, TDBC corresponds to the actual total number of signals that can be found in each of the four tested vehicles used for its evaluation, as the DBC files used for as ground truth were obtained from the actual OEM. In our case, we do not have a complete ground truth for each of our testing traces and, due to this limitation, we evaluate Phase 2 using the ratio CE/TDBC.

No status/multi-value signals were intentionally triggered by the driver at data collection time and, therefore, their bit flip cannot be possibly observed. Since such signals are typically 1–4 bit long, in the evaluation we consider only signals whose length is equal or greater than 5 bit.

Let S be the set of known signals in the ground truth. In general, a token t is considered to be correctly identified if exists a signal $s \in S$ such that the boundaries of t and s coincide, i.e., their start/end bits are the same. In our case, we allow a 4 bit tolerance error for the most significant bit in response to a widely known problem that affects tokenization [9], [12], [13]. For many signals – especially those representing physical information – it is very difficult to record flips in their most significant bits, as they can be activated only under exceptional (and often extreme) circumstances. For instance, the maximum theoretical speed of a car is hardly recorded, unless the car is driven on purpose to reach that speed. While no error can be tolerated in the identification of the start bit of a token, as it would inevitably affect the decoding of the scale factor and offset, we argue that a certain tolerance error is acceptable for the end bit. As a matter of fact, cutting off the most significant bits of a signal might not allow us to correctly interpret the maximum value that can be reached by the signal, as intended by the manufacturer. Nonetheless, it does not affect the interpretation of the signal when it carries physical values under "normal" driving conditions. We choose 4 bit as tolerance error, because it seems a good trade off between addressing the described issue and a fair evaluation of our algorithm.

We first perform an extensive tuning on 8800 combinations of the parameters α, β, and τ, in order to find the one that guarantees the best CE/TDBC ratio. Figure 7 presents the average results obtained on all tested vehicles with every combination of α and β, and $\tau = 0.3$. To be noted that we evaluated other values of τ as well, but present here only the best value. The results show that a maximum mean CE/TDBC score of 77.6 % is achieved with $\tau = 0.3$, $\alpha = 0.2$, and $\beta = 0.2$. This suggests that the number of long tokens constitutes the most valuable contribution for the likelihood score.

Figure 8 compares Phase 2 of CANMatch with READ [9] and the tokenization algorithm of LibreCAN [13], which, at the time of writing, are the start-of-the-art algorithms for tokenization. The results show that Phase 2 outperforms READ and LibreCAN on all considered vehicles of DVT. The difference in the performance, especially remarkable for some of the vehicles, is driven by the presence of LE cross-byte signals – signals whose information is encapsulated in multiple

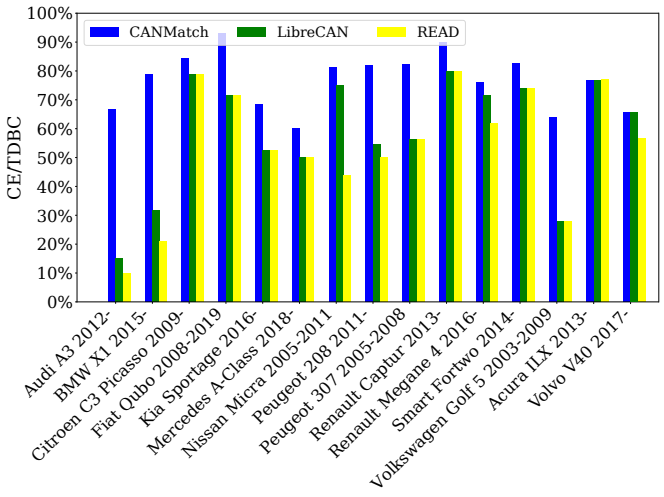

Figure 8. Phase 2 performance evaluation: comparison with state of the art.

Table II
ENDIANNESS IMPACT ON TOKENIZATION.

	Recall	Precision
Big Endian	85.2 %	99.2 %
Little Endian	60 %	100 %

(consecutive) bytes. The cross-byte signals are typically signals which carry physical information and, therefore, require a large number of bits to be represented, e.g. signals related to throttle pedal, steering wheel, engine etc. Failing to take into account the endianness during tokenization can result into splitting a cross-byte signal into two (or more) signals, as in the example illustrated in Figure 4.

Our analysis on SDDBC shows that, out of 4667 cross-byte frames, 65.3 % of the signals are represented according to the BE format and 34.7 % are represented according to the LE format. We also observed that all signals within a frame are represented with the same endianness. Our tokenization algorithm exploits this information to correctly identify the endianness of all cross-byte signals and, subsequently, their correct boundaries.

Table II shows how endianness impacts the performance of the tokenization process on DVT. On the one hand, Phase 2 is able to identify tokens represented in BE format better than tokens represented in LE format. On the other hand, all tokens labelled as LE are actually in LE format, while a small rate (0.8 %) of tokens labelled as BE are, instead, in LE format.

Phase 2 achieves a performance similar to LibreCAN and READ on ODVT. The reason is the absence of LE signals in this set, as reported in the generated ground truth available in [35]. It is still worth noting that, in this case, the search for LE signals does not impact negatively the capability of Phase 2 of correctly identifying BE signals. This further corroborates the findings presented in Table II.

D. Phase 3 Evaluation

We evaluate Phase 3 independently of Phase 1 and Phase 2. We extract the time series related to each signal present in every trace of DVT and ODVT according to the information contained in their DBC files.

The tuning of the different parameters of Phase 3 (presented in the following) is based on DVT. Then, for each vehicle, we partition the time series into two distinct subsets: Decoded Signals (*DS*) and Tokens (*T*). The former set contains signals emulating the results of the decoding process of Phase 1, while the latter corresponds to the time series of the tokens that emulate the results of the tokenization process of Phase 2. To analyze the performance for different number of signals decoded through Phase 1, we test Phase 3 assuming three different sizes of the *DS* set: 20, 50 and 80 %. As a consequence, we assume 80, 50 and 20 % for the size of *T*. We also partition the time series of *DS* and *T* randomly. To minimize the impact of this random choice on the evaluation, we repeat this selection 10 times.

The chosen metric for validating Phase 3 is the mean accuracy. In multi-label classification, accuracy is the ratio of the number of correctly classified samples to the total number of samples. In this case, a sample corresponds to a token in *T* to be translated. Let t be a token and *R* the set of all its redundant signals in *T*. We consider t correctly classified if:

- it is matched to any $s \in R$, if R is not empty;
- it is not matched to any signal in *DS*, if R is empty.

For this task, we consider three metrics that calculate the similarity between curves (time series): Area method, Root Mean Squared Error (RMSE) [37] and Dynamic Time Warping (DTW) [38]. The Area method measures the area of two curves in a 2-D space. RMSE corresponds to the mean of the squared differences between each couple of points of two vectors for every x-coordinate. DTW finds an optimal correspondence between two time series, through a non-linear distortion with respect to the independent variable (here, the time).

The value of the threshold θ used to determine whether a token is similar enough to a signal is chosen through an extensive tuning taking into account the specific similarity metric used to calculate the score. Figure 9 illustrates how the accuracy of Phase 3 varies according to θ. All the results show that starting from the lowest value of θ, the accuracy increases until it reaches the optimal value, then it starts decreasing. In fact, if θ is set too low, some tokens are not matched with their redundant signal. If it is set too high, tokens with no corresponding redundant signal in *DS* are wrongly matched to another signal in *DS*.

The results also show that the properties of DTW, i.e., being resilient to temporal shifts and distortions, do not constitute an advantage in this case. In fact, the temporal skew between different redundant signal within the CAN bus is minimal, i.e., in the order of ms. On the contrary, DTW is sensitively more time-costly compared to the other two metrics. According to this tuning, a maximum accuracy of 94.8, 93.6 and 92.9 % is achieved with RMSE with $\theta = 0.01$ on a *DS* set of, respectively, 20, 50 and 80 % of the ground truth.

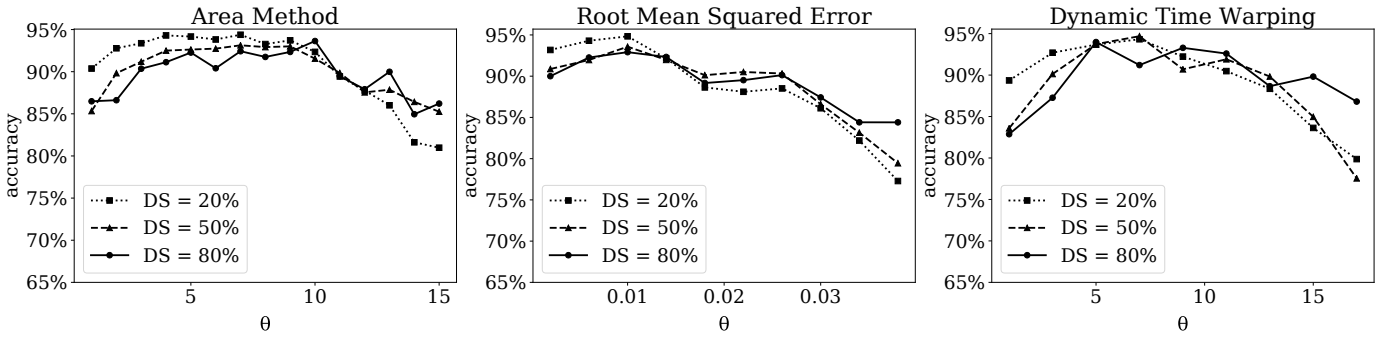

Figure 9. Tuning on the similarity threshold for Area method, RMSE and Discrete Frechet Distance (DFD) at the varying of the size of *DS* size.

1) Clustering evaluation: The next step is to understand if performing clustering first and then computing the similarity scores within each cluster can help decreasing the computation time without decreasing the accuracy. For the selection of the clustering algorithm, we consider the following properties: i) it should not require to specify in input the number of clusters. In fact, the number of vehicle functions that can be found in a specific vehicle is not known a priori, ii) it should not require to specify in input the spatial distribution of the data, as it is not known a priori, iii) it should handle noise, i.e. outliers, because not all tokens are redundant of known signals (this property also helps to further reduce the computational cost, as outliers are excluded from the computation of the similarity scores), and iv) the time complexity should be as low as possible. After taking into account a variety of clustering algorithms (e.g. centroid-based, hierarchical, distribution-based), we found out that density-based ones fit the aforementioned requirements. In particular, we focus on two most known density-based algorithms: Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [32] and Ordering Points To Identify the Clustering Structure (OPTICS) [39].

The hyperparameters of DBSCAN are ϵ, which corresponds to the maximum distance between two points to consider them neighbours, and *minPoints*, which corresponds to the number of neighbour samples to form a cluster. DBSCAN can find arbitrarily-shaped clusters of dense points, i.e., the cluster size is at least *minPoints*. Samples which lie in scarcely-dense areas are labelled as outliers. OPTICS is based on DBSCAN and uses the same parameters ϵ and *minPoints*. However, in case of OPTICS, ϵ can be ignored (or set to ∞), since it does not have any effect on the clustering accuracy [39].

In the following, we evaluate the impact of clustering on the vehicle function performance, by assuming RMSE as metric, with $\theta = 0.01$ and $DS = 50\,\%$. We set the parameter $minPoints = 2$. This is the minimum number of neighbour samples to form a cluster composed of at least one known signal and a token. We then tune ϵ for DBSCAN. The parameter ϵ should be set in a way that the outputted clusters i) should fit well to the vehicle functions in order to reduce the computational time, i.e. ideally there should be a ground truth signal representing only one vehicle function in each cluster, and ii) should be large enough so that all tokens representing the same vehicle functions are located in the same cluster.

Figure 10 illustrates three main metrics when varying ϵ: (i) mean number of clusters, (ii) mean number of outliers, and (iii) mean number of redundant signals in different clusters. It can be observed that the smaller the ϵ, the more samples are considered as noise (outliers). Also, the number of clusters initially increases with ϵ, as fewer samples are considered outliers, thus allowing the forming of new clusters. This growth is then followed by a decline, as the clusters become larger and incorporate samples previously belonging to multiple clusters. The figure also reports the number of redundant signals that are assigned to different clusters. As expected, when clusters are smaller on average, it happens more frequently that redundant signals are split into different clusters. The results reported in Figure 11 reflect the remarks made for Figure 10. For low values of ϵ, the mean accuracy of DBSCAN is inferior to the

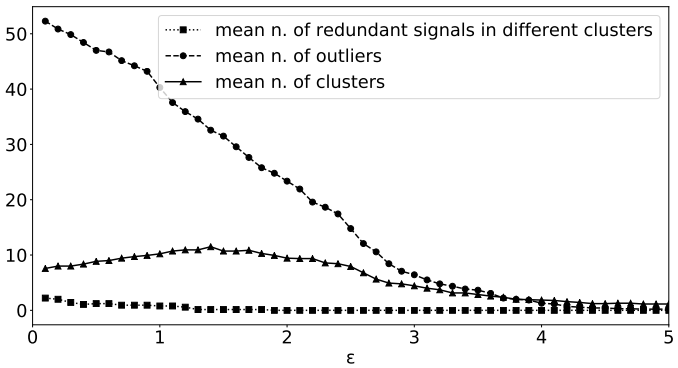

Figure 10. Tuning of parameter ϵ for DBSCAN.

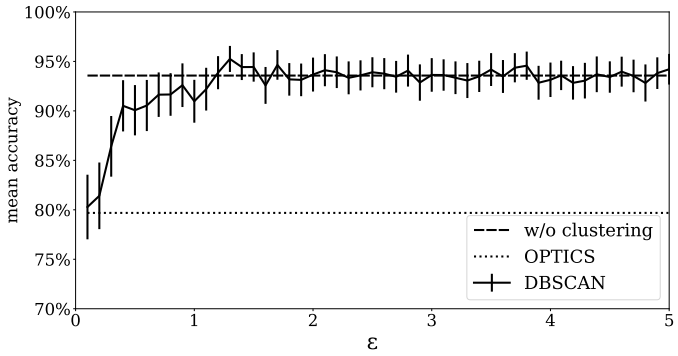

Figure 11. Comparison of the accuracy obtained for the translation of redundant signals between preliminary clustering through DBSCAN (tuned on ϵ), OPTICS and without clustering.

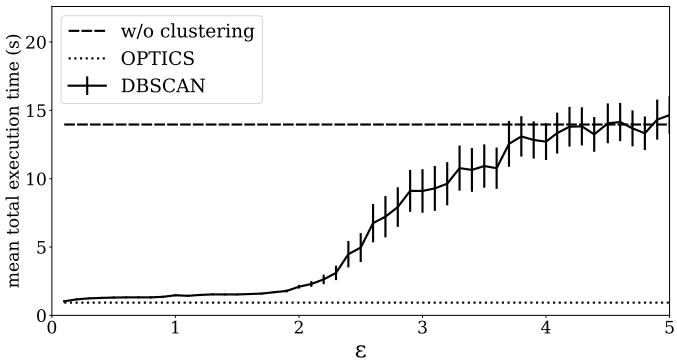

Figure 12. Comparison of the total execution time (ET) for the translation of redundant signals between preliminary clustering through DBSCAN (tuned on ϵ), OPTICS and without clustering.

maximum value achieved without clustering. Vice versa, when the clusters are large enough, all the similarity scores relevant to identify all the redundant tokens are calculated.

Figure 12 compares the total execution time (*ET*), defined as the sum of time required for clustering (if performed) and time for the computation of all similarity scores, of DBSCAN, OPTICS and without preliminary clustering. The figure highlights that *ET* increases with ϵ in case of DBSCAN. In fact, the bigger the clusters are on average, the more similarity scores have to be calculated. For low values of ϵ (≤ 2), the *ET* using DBSCAN and OPTICS is one order of magnitude inferior compared to the baseline approach without clustering, while for high ϵ (≥ 4), *ET* is equal or higher.

Taking into consideration the results shown in Figures 11 and 12, we conclude that the optimal trade off between performance and total execution time for Phase 3 is obtained by performing clustering with DBSCAN and $1 < \epsilon < 2$.

2) Format decoding: After identifying the vehicle function of each redundant token, the scale factor and offset are decoded through linear regression, in the way described in Section III-C. Let S_c be the time series of the target signal S parsed according to the calculated format, and S_o the time series of S parsed according to the original format, as defined in its ground truth. Let Normalized Root Mean Squared Error (NRMSE) be the RMSE between S_c and S_o, normalized by the range of values in S_o:

$$NRMSE = \frac{\text{RMSE}(S_c, S_o)}{\max(S_o) - \min(S_o)} \tag{8}$$

We evaluate the accuracy format extracted for S through Format Decoding Accuracy (FDA), defined as:

$$FDA = 1 - NRMSE \tag{9}$$

We evaluate the FDA of the redundant signals of each vehicle in DVT and ODVT through a leave-one-out-cross-validation approach. Each signal is iteratively considered as the signal of reference, and all its redundant signals as the tokens whose vehicle function has been decoded through the similarity score and whose format is still unknown. Figure 13 shows the mean FDA obtained for each vehicle in DVT and ODVT. The figure shows that the performance of Phase 3 is consistent among different vehicles and different OEMs.

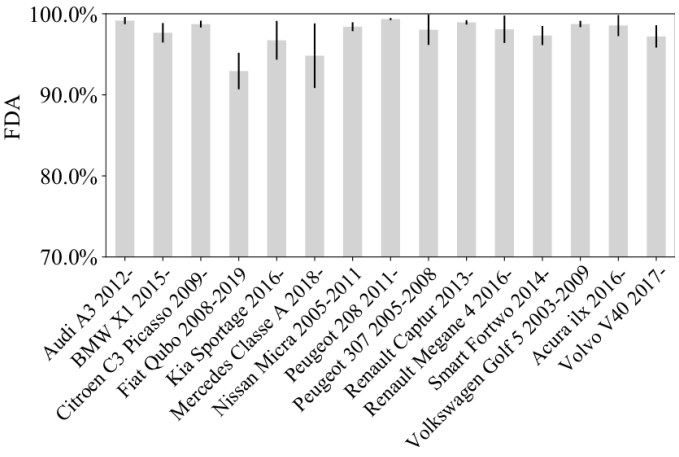

Figure 13. Phase 3 performance evaluation: mean FDA for each vehicle in DVT and ODVT.

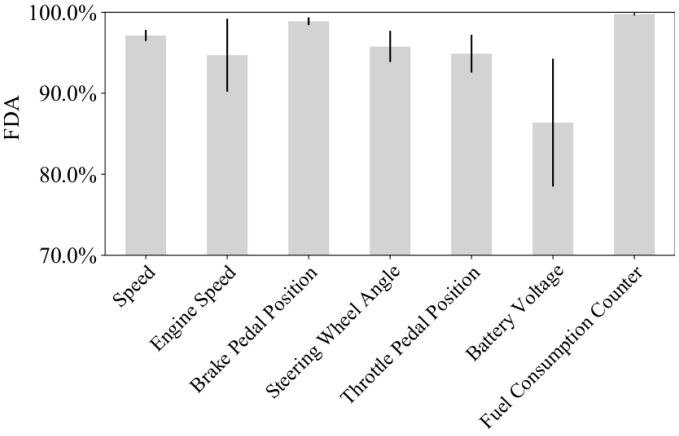

Figure 14. Phase 3 performance evaluation: mean FDA for each vehicle function represented by redundant signals in DVT and ODVT.

We also evaluate the accuracy of the format decoding by vehicle function, as reported in Figure 14. The presented vehicle functions are those for which redundant signals are present in the ground truth of SDDBC and OpenDBC. As mentioned in Section III-C, the *speed* group is composed of the vehicle speed and the wheel speed signals. Figure 14 highlights that there is a moderate variability in the performance of Phase 3 on different vehicle functions.

E. CANMatch Framework Evaluation

We validate CANMatch using the DVT and ODVT sets and test the whole pipeline assuming different sizes of SDDBC.

Similarly to Phase 1, the vehicles composing each subset are chosen randomly. This selection is performed 10 times, to minimize the impact of this random choice on the overall evaluation. Each vehicle in the *DVT* and *ODVT* sets is tested once when assuming all DBC in our possession (minus the one currently under examination) in the ground truth set, and 10 times when considering a ground truth of smaller size.

Figure 15 illustrates the aggregated performance obtained for each ground truth size. We choose two metrics that show the completeness and the reliability of the translation of CAN

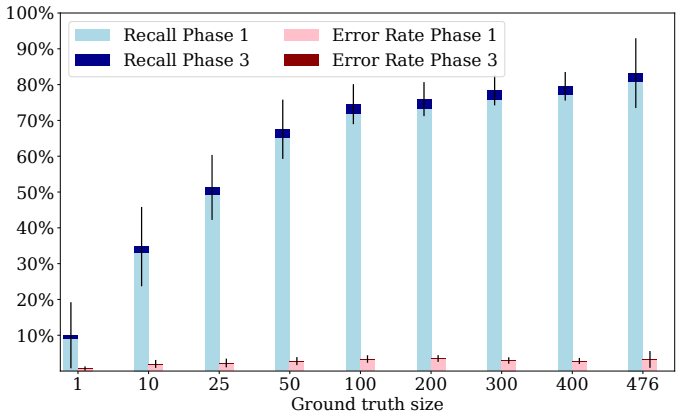

Figure 15. Recall and error rate for different sizes of the ground truth dataset.

signals achieved by CANMatch. In particular, since we want to highlight the cumulative contribution of each translation phase (i.e., 1 and 3) to the reverse engineering process, we employ the error rate metric (i.e., rate of incorrectly translated signals) instead of precision. The figure highlights that the majority of signals are decoded through Phase 1. As only 13.1 % of the signals in the DVT are redundant, and given that most of them are also decoded through Phase 1, only few of them are left to be decoded by Phase 3. The figure also shows that the overall rate of signals which are wrongly decoded is almost irrelevant (especially those obtained through Phase 3), confirming the high precision of CANMatch.

It is also worth noting that the confidence interval is maximum when the ground truth includes only one vehicle and shrinks with the increase of the ground truth size, with the exception of the whole ground truth, for which the confidence interval is affected by the lower number of tests conducted. This confirms that the more comprehensive data we have in our ground truth set, in terms of quantity and diversity, the more reliable CANMatch becomes.

Table III reports the mean total execution time required by each step of the pipeline for each vehicle trace. The table shows high variance in the execution time among the different vehicles. The factors that seem to affect the computational load seem to be the year of release of the vehicle and the market segment. Indeed, latest and/or high-end vehicle models are usually equipped with more ECUs compared to older and/or cheaper ones. More ECUs means more frames to decode, hence longer time needed for the reverse engineering.

With a total computational time well below one minute for most of the tested vehicles (obtained with modest hardware resources), we conclude that CANMatch is the fastest tool for CAN bus reverse engineering.

V. Discussion

CANMatch offers a plug-and-play solution that, with minimum hardware equipment (a CAN dongle) and minimum amount of data (1-2 minutes of CAN raw trace) related to an unknown vehicle, is able to reverse engineer it in few seconds. However, our tool makes intensive use of DBC files which, if not acquired from third parties, implies that a (manual) reverse engineering process has to be previously put in place for a number of vehicles. While the data collection for the ground truth is certainly time consuming, the more vehicles are reverse engineered, the faster and more accurate the process becomes.

On the contrary, other solutions require the same steps to be performed for any number of vehicles to reverse engineer. This includes the installation of external sensors for the gathering of GPS/IMU data, injection of diagnostic messages through the OBD-II port and manual actions within the vehicle. For these reasons, CANMatch is consistently more scalable than other tools for automated CAN bus reverse engineering. A combined use of our tool with other automated solutions would guarantee the fastest reverse engineering overall. A number of vehicles can be initially reverse engineered with semi-automated tools, such as LibreCAN [13], in shorter time compared to the manual approach, until the ground truth is large enough to ensure the desired performance with CANMatch, which can then be used to significantly decrease time and effort of the overall reverse engineering process.

However, speeding up the automated reverse engineering process raises non-negligible security concerns. Vehicles are becoming more and more interconnected to each other and to the infrastructure. We can also expect an increase of ECUs that will be equipped with a wireless interface [40], thus expanding the access surface to be exploited by potential adversaries for remote attacks. With CANMatch, an adversary with the ability to compromise one or few popular wireless-enabled ECUs could reverse engineer plenty of vehicles within minutes (e.g. at a busy road intersection). The relative ease of access to a high number of vehicle CAN formats would encourage attackers to design large-scale attacks. Furthermore, an adversary could directly reverse engineer and inject a pre-designed vehicle model-agnostic attack in the same time frame, thus cutting off the necessity of extracting information on the target vehicle model prior to the attack event.

Since CANMatch needs a clear reading of the CAN IDs to function, if car manufacturers intend to provide better security and safety for their vehicles, they should obscure such information. For instance, they could abandon the practice of CAN frame ID reuse, i.e. each newly release vehicle model is equipped with a new set of CAN IDs. This solution does not imply any modification of the CAN protocol nor disruptions in the supply chain, as the same ECUs could be reused, but set to send frames with different IDs. However, related work suggests that frames might still be recognizable despite their ID by exploiting features such as the dynamicity of the payload or the mean sending frequency[1]. In this case, it would be sufficient to apply a preliminary step to associate the frames of the target vehicle with those known and, subsequently, use CANMatch. A more sound approach to secure the CAN data is the adoption of encryption. A number of encryption solutions proposed in literature have achieved convincing results in terms of security while being compliant with the physical limits of the bus [41]. In particular, AUTOSAR's module Secure Onboard Communication (SecOC), built to prevent replay

[1]The results of this study have been submitted to an IEEE conference and are currently under review. They are available upon request.

Table III
MEAN EXECUTION TIME FOR THE DIFFERENT STEPS OF THE PIPELINE ACCORDING FOR EACH VEHICLE IN THE DVT SET.

Vehicle	**Time (s)**						
	Phase 1		**Phase 2**	**Phase 3**			**Total**
	Trace Parsing	Matching	Tokenization	Clustering	Translation	Format Decoding	
Audi A3 2012-	5.1	12.5	24.8	1.2	4.0	0.1	47.6
BMW X1 2015-	3.3	14.4	24.7	0.9	9.8	0.1	53.2
Citroen C3 Picasso 2009-	5.3	5.8	4.5	0.2	1.1	0.1	17.1
Fiat Qubo 2008-2019	5.1	3.1	5.1	0.2	0.7	0.1	14.3
Kia Sportage 2016-	4.3	9.6	14.7	0.9	3.7	0.1	33.3
Mercedes A-Class 2018-	3.7	15.0	20.6	0.9	8.9	0.1	49.1
Nissan Micra 2005-2011	4.6	4.7	4.8	0.2	0.9	0.1	15.3
Peugeot 208 2011-	9.2	9.4	11.7	0.4	1.8	0.1	32.6
Peugeot 307 2005-2008	2.2	6.0	4.9	0.3	1.1	0.1	14.5
Renault Captur 2013-	12.3	7.5	13.0	0.4	1.8	0.1	35.1
Renault Megane 4 2016-	5.1	7.9	13.7	0.5	1.5	0.1	28.7
Smart Fortwo 2014-	11.3	8.9	17.7	0.4	1.3	0.1	39.7
Volkswagen Golf 5 2003-2009	3.0	8.1	12.3	0.7	2.8	0.1	27.0
Acura ILX 2013-	5.2	6.0	10.7	0.6	1.4	0.1	24.0
Volvo V40 2017-	4.5	9.2	12.0	0.8	1.6	0.1	28.2
Mean	5.6	8.6	13.0	0.5	2.8	0.1	30.6

attacks, tampering and spoofing through ECU authentication and frame encryption, would make a reverse engineering approach based on frame matching much more difficult to execute [42].

VI. CONCLUSION

In this paper, we present CANMatch – an automated CAN bus reverse engineering framework that exploits the reuse of CAN frame IDs among vehicle models. To the best of our knowledge, CANMatch is the least-invasive and minimal approach in terms of employed hardware, manual effort, and execution time on the vehicle to reverse engineer. It requires in input the raw CAN trace to be decoded and a ground truth dataset of DBC files, necessary for the identification and decoding of signals in the initial phase.

CANMatch includes also a tokenization algorithm, which, differently from other state-of-the-art solutions, considers the endianness of the extracted tokens. This algorithm is validated against other state-of-the-art solutions, showing equal or higher accuracy on all tested vehicles. In addition, CANMatch proposes a method to identify redundant signals by exploiting a combination of density-based clustering and computation of similarities between tokens and previously-decoded signals.

We validated our solution on a diverse dataset of real CAN traces collected from 479 parked vehicles and 15 moving vehicles, obtaining comparable results on all of them. This advocates for its universal usage.

Future work includes additional optimization for each step of the CANMatch pipeline to further reduce the overall computation time and increase accuracy. In addition, we intend to conduct a more comprehensive investigation upon the CAN format design choices for individual OEMs (e.g. priority and sending frequency of the frames). The goal is to discover hidden patterns that can be exploited to identify the sending ECUs and to use this information to decode the content of the frames, thus reducing the need for DBC files.

ACKNOWLEDGEMENT

We acknowledge support from the National Research Fund (FNR) under grant number PRIDE15/10621687. We thank Xee/Eliocity SAS for the provided datasets and support we used to design and validate our solution.

REFERENCES

[1] S. Jafarnejad, "Machine Learning-based Methods for Driver Identification and Behavior Assessment: Applications for CAN and Floating Car Data," Ph.D. dissertation, University of Luxembourg, Esch-sur-Alzette, Luxembourg, 2020.

[2] U. Fugiglando, E. Massaro, P. Santi, S. Milardo, K. Abida, R. Stahlmann, F. Netter, and C. Ratti, "Driving behavior analysis through CAN bus data in an uncontrolled environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 2, pp. 737–748, 2018.

[3] L. Nkenyereye and J.-W. Jang, "Integration of big data for querying CAN bus data from connected car," in *9th International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 946–950.

[4] M. Bertoncello, G. Camplone, P. Gao, H.-W. Kaas, D. Mohr, T. Möller, and D. Wee, "Monetizing car data—new service business opportunities to create new customer benefits," *McKinsey & Company*, 2016.

[5] SAE, "Recommended practice for a serial control and communications vehicle network," *SAE J1939 Standards Collection*, 2010.

[6] C. Quigley, D. Charles, and R. McLaughlin, "CAN Bus Message Electrical Signatures for Automotive Reverse Engineering, Bench Marking and Rogue ECU Detection," in *SAE Technical Paper*, SAE International, Apr. 2019.

[7] M. Jaynes, R. Dantu, R. Varriale, and N. Evans, "Automating ECU identification for vehicle security," in *15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2016, pp. 632–635.

[8] M. Markovitz and A. Wool, "Field classification, modeling and anomaly detection in unknown CAN bus networks," *Vehicular Communications*, vol. 9, pp. 43–52, 2017.

[9] M. Marchetti and D. Stabili, "READ: Reverse engineering of automotive data frames," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1083–1097, 2018.

[10] B. C. Nolan, S. Graham, B. Mullins, and C. S. Kabban, "Unsupervised time series extraction from controller area network payloads," in *IEEE 88th Vehicular Technology Conference (VTC-Fall)*, IEEE, 2018, pp. 1–5.

[11] T. Huybrechts, Y. Vanommeslaeghe, D. Blontrock, G. Van Barel, and P. Hellinckx, "Automatic reverse engineering of CAN bus data using machine learning techniques," in *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Springer, 2017, pp. 751–761.

[12] M. Verma, R. Bridges, and S. Hollifield, "ACTT: Automotive CAN tokenization and translation," in *International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2018, pp. 278–283.

[13] M. D. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. G. Shin, "LibreCAN: Automated CAN Message Translator," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, ACM, 2019, pp. 2283–2300.

[14] A. Buscemi, G. Castignani, T. Engel, and I. Turcanu, "A Data-Driven Minimal Approach for CAN Bus Reverse Engineering," in *3rd IEEE Connected and Automated Vehicles Symposium (CAVS)*, Victoria, Canada: IEEE, Oct. 2020.

[15] U. Ezeobi, H. Olufowobi, C. Young, J. Zambreno, and G. Bloom, "Reverse Engineering Controller Area Network Messages using Unsupervised Machine Learning," *IEEE Consumer Electronics Magazine*, 2020.

[16] V. H. Le, J. den Hartog, and N. Zannone, "Security and privacy for innovative automotive applications: A survey," *Computer Communications*, vol. 132, pp. 17–41, 2018.

[17] J. Cui, L. S. Liew, G. Sabaliauskaite, and F. Zhou, "A review on safety failures, security attacks, and available countermeasures for autonomous vehicles," *Ad Hoc Networks*, vol. 90, p. 101 823, 2019.

[18] K. B. Kelarestaghi, M. Foruhandeh, K. Heaslip, and R. Gerdes, "Intelligent transportation system security: impact-oriented risk assessment of in-vehicle networks," *IEEE Intelligent Transportation Systems Magazine*, 2019.

[19] G. Brindescu. (2015). "DARPA Hacked a Chevy Impala Through Its OnStar System," [Online]. Available: https://www.autoevolution.com/news/darpa-hacked-a-chevy-impala-through-its-onstar-system-video-92194.html (visited on 04/02/2021).

[20] S. Jafarnejad, L. Codeca, W. Bronzi, R. Frank, and T. Engel, "A car hacking experiment: When connectivity meets vulnerability," in *2015 IEEE Globecom Workshops (GC Wkshps)*, IEEE, 2015, pp. 1–6.

[21] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, no. S 91, 2015.

[22] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11–25, 2011.

[23] W. Wu, R. Li, G. Xie, J. An, Y. Bai, J. Zhou, and K. Li, "A survey of intrusion detection for in-vehicle networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2019.

[24] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, Austin, TX, 2016, pp. 911–927.

[25] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PloS one*, vol. 11, no. 6, 2016.

[26] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "VoltageIDS: Low-Level Communication Characteristics for Automotive Intrusion Detection System," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2114–2129, 2018.

[27] International Organization for Standardization, "Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," ISO 11898-1, Dec. 2015.

[28] *Intel IXP4XX Product Line of Network Processors and IXC110 Control Plane Processor: Understanding Big Endian and Little Endian Modes*, 254237-001, Intel, Dec. 2003.

[29] A. Blin. (). "CAN bus reverse-engineering with Arduino and iOS," [Online]. Available: https://medium.com/@alexandreblin/can-bus-reverse-engineering-with-arduino-and-ios-5627f2b1709a (visited on 08/10/2021).

[30] C. Smith. (). "The Car Hacker's Handbook: A Guide for the Penetration Tester," [Online]. Available: https://publicism.info/engineering/penetration/6.html (visited on 08/10/2021).

[31] Vector. (). "Managing Network and Communication Data with CANdb++," [Online]. Available: https://www.vector.com/int/en/products/products-a-z/software/candb/ (visited on 05/27/2020).

[32] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, 1996, pp. 226–231.

[33] W. H. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of classification*, vol. 1, no. 1, pp. 7–24, 1984.

[34] W. Choi, S. Lee, K. Joo, H. J. Jo, and D. H. Lee, "An Enhanced Method for Reverse Engineering CAN Data Payload," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3371–3381, 2021.

[35] C. AI. (). "OpenDBC," [Online]. Available: https://github.com/commaai/opendbc (visited on 08/05/2021).

[36] PEAK Systems. (), [Online]. Available: https://www.peak-system.com/PCAN-USB-FD.365.0.html?&L=1/ (visited on 01/13/2021).

[37] D. Wackerly, W. Mendenhall, and R. L. Scheaffer, *Mathematical statistics with applications*. Cengage Learning, 2014.

[38] T. Eiter and H. Mannila, "Computing discrete Fréchet distance," Citeseer, Tech. Rep., 1994.

[39] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.

[40] J. Kleylein-Feuerstein, F. Joas, and R. Steinhilper, "Remanufacturing of Electronic Control Units: An RFID Based (Service) Interface," *Procedia CIRP*, vol. 29, pp. 168–172, Dec. 2015.

[41] O. Avatefipour and H. Malik, "State-of-the-Art Survey on In-Vehicle Network Communication (CAN-Bus) Security and Vulnerabilities," *CoRR*, vol. abs/1802.01725, 2018. eprint: 1802.01725. [Online]. Available: http://arxiv.org/abs/1802.01725.

[42] Autosar. (). "Specification of Secure Onboard Communication," [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/20-11/AUTOSAR_SWS_SecureOnboardCommunication.pdf (visited on 11/30/2020).