Evaluating Feature Encodings for Unsupervised Machine Learning Classification in Automotive Ethernet Network

Kumar Ashutosh Anand*†, Stefanie Merz*†, Michael Heigl†, Dalibor Fiala*§, Hannes Schulz‡, Wolfgang Kirmair‡, and Martin Schramm†
*University of West Bohemia, Pilsen, Czech Republic
{kanand, stmerz, dalfia}@kiv.zcu.cz
†Deggendorf Institute of Technology, Deggendorf, Germany
{kumar.anand, stefanie.merz, michael.heigl, martin.schramm}@th-deg.de
‡BMW Group, Munich, Germany
{Hannes.Schulz,Wolfgang.Kirmair}@bmw.de
§CCA Group, Pilsen, Czech Republic

***Abstract*—Categorical attributes such as MAC and IP addresses constitute an integral part of Ethernet network data, and play a crucial role in modern network infrastructure. Representing these intrinsic entities with high cardinality presents a considerable performance challenge pertaining to machine learning tasks. In order to better manage the representations of the categorical attributes found in network data, this work presents new methods for transforming them. Some of these encoding schemes are designed using domain knowledge to limit the number of dimensions introduced in data while performing transformations. This study uses two specific *Autoencoder* deep neural networks for the unsupervised classification task to help assess the classification performance for the proposed encoding schemes. These varied encodings used to transform Ethernet network data from a real vehicle serve as a novel contribution to the feature engineering for analyzing the network data using machine learning approaches. The evaluation results show that the proposed techniques have a key impact on the classification performance, and the encoding schemes *IE* and *ISF* performed reasonably well in all three attack scenarios for each model.**

***Index Terms*—Feature Encoding, High Cardinality, Anomaly Detection, Automotive Ethernet, Categorical Attributes, Unsupervised machine learning.**

I. Introduction

In the evolving landscape of cybersecurity, anomaly detection have already started gaining traction for enhancing security and safeguarding network solutions and infrastructure. By spotting deviations from the normal network behavior, these systems flag anomalous events happening in the network infrastructure. Since Ethernet network technology [1] has seen its steady proliferation in the automotive industry due to higher connectivity demand, it would be only relevant to adapt and bring such solutions to the automotive domain.

Pertaining to the reasons such as increased bandwidth demand, incorporation of modern features like advanced driver assistance systems etc., the automotive network heavily relies on data communication between various internal units. For this reason, data-driven paradigm could appear as a suitable approach to design anomaly detection systems for automotive network, permitting them to determine the patterns and behaviors accurately through the data it encounters [2].

When developing such data-driven solutions, feature engineering holds paramount significance, as it serves as a crucial intermediate step between raw data and models in a machine learning (ML) pipeline. The transformation of raw network data into relevant attributes through feature engineering enhances its significance, enabling machine learning models to learn more effectively [3].

In the feature engineering process, feature encoding (FE) is a step where categorical features are transformed into numerical values that can be efficiently used for ML models. This step could prove crucial because the choice of an encoding technique can significantly influence the performance of an ML-based application [4]. Therefore, selecting an appropriate feature encoding technique could enable the model or classifier to distinguish between normal and anomalous behavior with higher accuracy [5].

The key contributions and goals that are aimed to be achieved through this publication are listed as below.

- Feature encodings for categorical data attributes, which could be used for batch and streaming data in an unsupervised setting.
- Utilizing standard network domain knowledge for transforming and representing categorical attributes such as IP address, MAC address through proposed encoding mechanisms.
- Performance comparison among two versions of the state-of-the-art classifier *Autoencoder* model, presented in Section VI with different feature encodings based on real automotive Ethernet network data.

The remainder of this work is structured as follows: Section II presents the state-of-the-art and related work to provide the relevant background to the reader about the existing feature encodings. Section III deals with our proposed work of feature encodings. The Section IV presents our evaluation framework for evaluation of proposed encodings. The Section V discusses the metrics used for evaluating the classification performance. The Section VI explains our experimental evaluation for feature encodings with models in the evaluation framework and covers a breakdown of our methodology. The Section VII provides the discussion about the outcomes achieved with the proposed encodings. Finally, in Section VIII, we present the conclusions about our current work and decide the directions and prospects for future work.

II. Related Work

As the ML algorithms slowly started gaining ground for classification purposes in the last decades, many research studies have been undertaken towards designing anomaly detection systems to detect malicious activities in the network infrastructure in order to enhance security. In recent years, some studies were also aimed at suggesting and devising techniques to handle categorical or non-numerical features in order to make them compatible with ML models. One such research work was performed in [5] by Shao, where the focus predominantly remained on the encoding of IP addresses for making network intrusion detections. The three methods of encoding IP addresses such as converting them to binary numbers, splitting IP address into four octets, and one-hot encoding (OHE) were applied to detect intrusions in the network traffic. The study found the split IP encoding technique to be better performing than binary IP and OHE [5]. It would be worth mentioning here that OHE is quite a popular way for handling categorical feature values due to its simplicity and are already available in popular data manipulation libraries like pandas [6] and machine-learning libraries such as scikit-learn [7]. The CatBoost library [8] bases its mechanism of transforming the categorical attributes into numerical ones on the prior knowledge of the labels or targets in terms of target statistics. Therefore, its application is perceived to be limited only to a supervised setting. Furthermore, the target statistics-based encoding schemes could potentially introduce target leakage, which could, in turn, lead to over-fitting issues in the model's generalization [9]. In paper [9], Johnson and Khoshgoftaar found that when the categorical features are encoded with approaches other than OHE encoding, their inclusion significantly improved the classification performance for all ensemble learners used like XGBoost. Although the study has been based on the evaluation of medical data, it assesses the impact of OHE encoding on categorical attributes and observed that it is not suitable for high-cardinality categorical features, as it achieved little-to-no improvement in the classification performance. Hence, OHE encoding would not be included in our evaluation experiments. Another potential candidate for handling categorical attributes, particularly aimed at handling IP addresses and ports in network data, was introduced in the paper [10]. The primary idea presented through this paper is to determine the similarities between IP addresses by using an approach similar to Word2vec [11] in a network data context. In this IP2Vec approach, IP address contexts are extracted from flow-based data, and the training samples are formed for training the neural network. Once the training is complete, the weights of the hidden layer could be used as a contextual feature representation for the IP addresses. However, this possible encoding scheme was not considered further in our evaluation because of the following limitations:

- As dynamic network behavior could impact the context generation for IP addresses, the feature embeddings, thus generated, could be suboptimal.
- As a neural network is trained to generate these feature embeddings, it may require some hyperparameter tuning for the neural network to generate optimal feature embeddings. Generally, this hyperparameter tuning is exercised at a later stage to optimize any model's classification performance.
- Similarly to the OHE encoding technique, it could be assumed that there would not be any explicit vector representations for IP addresses found in test data, which are not already seen beforehand in the training data.

Moreover, no previous work, to our knowledge, has been found to consider or recommend any encodings for the MAC addresses, i.e., layer 2 network information in the data, when used with ML algorithms. This lack of layer 2 information could also result in some information loss from network data, which could be helpful for ML models to generalize data better and improve their performance accuracy for handling layer-2 attacks, particularly when developing such solutions for a target network environment.

So, keeping these limitations in mind, we identified some requirements for feature encoding schemes to fulfill the criteria to be applied in real-world network data scenarios.

R01 - As already pointed out in the Section I, the ground truth (i.e., target, class, labels) being scarce, it demands the encoding scheme(s) to be able to operate in an unsupervised way.

R02 - The encoding schemes should be capable of handling the encoding process performed on evolving streaming data in dynamic network setting(s).

R03 - Another operation-related requirement would favor the non-existence of the prerequisite of dealing with hyperparameters to generate the representation or embeddings to replace categorical attribute values, i.e., no hyperparameters should be required for this data transformation step.

R04 - The encoding mechanism should, by design, prevent or limit the increase in the cardinality of the data being worked upon. For example, encoding measures such as OHE could lead to an explosion in the cardinality of highly dynamic network data due to the inherent working mechanism which could, in turn, weaken the performance of any ML model [9].

R05 - Another key requirement would be the inherent capability to work with unique categorical feature value(s) that were earlier never seen or met in the data encountered previously for training the model.

III. Feature Encodings

One of the prime aspects of building a data pipeline to realize any performant ML model is to handle categorical attributes coming with the data effectively. Network data primarily consists of MAC addresses, IP addresses, and protocol(s), etc., which are mainly considered as categorical attributes. These categorical features are assumed to undergo some form(s) of transformation and are finally turned into their equivalent numeric values, which could be used for model training and evaluation.

As already referenced in Section II, OHE and split IP encoding could be assumed as commonly used encoding techniques for categorical attributes for network data, as these could be used in an unsupervised setting. Another intuitive prospect for these transformations would be *Integer encoding*. Although the origin of integer encoding is hard to find in any scientific paper, this seems to be a first go-to approach for encoding MAC and IP addresses directly into corresponding numerical values by leveraging the *ipaddress*[1] Python package. Hence, this encoding has been included in our experiments for its performance evaluation. In addition, split IP encoding has been extended to transform MAC addresses, splitting them into six additional features, which is termed as *Octet Splitting* going forward, for the sake of simplicity.

In this work's scope, the proposed feature encoding techniques are listed here. To the best of our knowledge, these encodings, with respect to their distinct working mechanism, have not been found to be explored in any published work.

- *Hash Encoding*
- *Integer Scale Factor*
- *Hashed Scale Factor*
- *Integer Value Difference*
- *Logarithm Value Difference*
- *Hashed Value Difference*
- *NetMask_Int_HostMask_Int*
- *NetMask_Int_HostMask_Int*

A brief description of all the encoding techniques being incorporated in our evaluation framework (in addition to the proposed ones) could be found here:

Integer Encoding (IE): The integer encoding simply converts all data values in a given categorical feature or column into their equivalent integer values. For example, a given IP address, denoted by *ip*, could be transformed as in (1).

$$int_{ip} = int(ip) \tag{1}$$

In the same way, MAC addresses are converted into their equivalent integer values.

Hash Encoding (HE): The hash encoding technique transforms the given feature values into SHA256 hash values. The objective of this encoding is to introduce more separation (randomness) in closely-spaced values (e.g., consecutive IP addresses) and introduce anonymity into the data. In this encoding technique, the MAC addresses and IP addresses are hashed, and the hash values are further converted to their equivalent integer values to replace their original values in the respective columns in our implementation. For example, a given IP address could be modified as shown in (2).

$$HE_{ip} = int(hash(ip)) \tag{2}$$

It is worth noting here that this hash encoding technique has a distinct working mechanism as compared to the hash encoding available with Category Encoders library [2].

Octet Splitting (OS): By convention, IP addresses are structured in the form of a predefined number of octets, depending upon their protocol version. Similarly, MAC addresses also follow a specific octet structure, where the first three octets denote the manufacturer. This encoding technique bifurcates each octet residing in MAC and IP addresses as an individual feature and places their integer equivalent into the newly added column(s) in the data. The original four columns containing MAC (source, destination) and IP (source, destination) addresses are dropped eventually in this encoding.

Integer Scale Factor (ISF): In this encoding technique, the categorical features such as MAC and IP addresses are first converted into their integer equivalents and then transformed into a factor or fraction against the integer equivalents for the highest possible values for the feature(s) respectively. For example, a given IP address could be transfigured as in (3).

$$ISF_{ip} = \frac{int(ip)}{int(max(ip))} \tag{3}$$

The columns containing MAC addresses and IP addresses are encoded using this technique, and their original column values are substituted with derived values.

Hashed Scale Factor (HSF): Hashed Scale Factor encoding is accomplished similarly to Integer Scale Factor. Here, the MAC and IP addresses are hashed first with SHA256 hashing algorithm and converted to their integer equivalents. Then, the transformed value is taken as a fraction against the integer equivalent of the highest possible SHA256-hashed value. A given IP address could be substituted as in (4).

$$HSF_{ip} = \frac{int(hash(ip))}{int(max(sha256value))} \tag{4}$$

The MAC and IP addresses are encoded using this technique, and their original columns are dropped.

Integer Value Difference (IVD): Integer Value Difference is implemented to take the account of the absolute difference in the integer equivalents of the source MAC/IP and destination MAC/IP or any other possible pair of features that could

[1] https://docs.python.org/3/library/ipaddress.html

[2] https://contrib.scikit-learn.org/category_encoders/

be used to establish a relationship between them. This new generated value is then used as an additional feature for the data. For example, a given pair of source IP address and destination IP address could be used to assign a new column and determine its values, as in (5).

$$IVD_{ip} = abs(int(ip_{src}) - int(ip_{dst})) \tag{5}$$

Logarithm Value Difference (LVD): This encoding scheme transforms the value thereby achieved from the IVD encoding on the logarithmic scale. A given pair of source IP address and destination IP address could be processed as in (6).

$$LVD_{ip} = log_{10}(abs(int(ip_{src}) - int(ip_{dst}))) \tag{6}$$

Hashed Value Difference (HVD): Similar to IVD encoding, this encoding technique derives an additional feature as the absolute difference between the integer equivalents of the hash values of the source MAC/IP and destination MAC/IP. For example, a given pair of source IPs and destination IP addresses could be utilized as in (7).

$$HVD_{ip} = abs(int(hash(ip_{src})) - int(hash(ip_{dst}))) \tag{7}$$

For *IVD*, *LVD* and *HVD* encoding schemes, the MAC, and IP addresses are used to derive their respective new columns, and their original column are retained with their corresponding encoded values.

NetMask_Int_HostMask_Int (NIHI): This encoding technique is based on the principle that the IP address is a combination of the network mask and host mask or identifier. So, here the source and destination IP addresses are divided into their respective network mask and host mask, and their integer equivalents are taken as additional features in the data as in (8).

$$NIHI_{ip} = int(NetMask) + int(HostMask) \tag{8}$$

NetMask_Hash_HostMask_Hash (NHHH): In this encoding scheme, the idea is very similar to NIHI. The only difference here lies in the fact that both the network mask and host mask are hashed with SHA256 algorithm, and its integer equivalent is introduced as new columns in the data according to (9).

$$NHHH_{ip} = int(hash(NetMask)) + int(hash(HostMask)) \tag{9}$$

In regard to encoding schemes such as *NIHI* and *NHHH*, the IP address columns (source/destination) are encoded using (8) and (9) above, respectively, to constitute four new columns, and their original columns are dropped. However, MAC addresses are converted into their integer equivalents.

Table I represents which encodings fulfill the requirements listed out at the end of Section II. It is evident from the table that the proposed encodings are designed or extended to fulfill all the requirements specified, as compared to the popular OHE encoding scheme and IP2vec encoding.

TABLE I
REQUIREMENT LIST FOR ENCODING TECHNIQUES

FE	R01	R02	R03	R04	R05
IE	✓	✓	✓	✓	✓
HE	✓	✓	✓	✓	✓
OS	✓	✓	✓	✓	✓
ISF	✓	✓	✓	✓	✓
HSF	✓	✓	✓	✓	✓
IVD	✓	✓	✓	✓	✓
LVD	✓	✓	✓	✓	✓
HVD	✓	✓	✓	✓	✓
NIHI	✓	✓	✓	✓	✓
NHHH	✓	✓	✓	✓	✓
OHE	✓	✓	✓	×	×
IP2Vec	✓	×	×	✓	×

[a] The tick mark (✓) indicates the fulfillment of the requirement by each encoding technique. In contrast, the cross mark (×) represents otherwise.

IV. EVALUATION FRAMEWORK

Our evaluation framework includes the phases and steps needed to perform end-to-end ML-based anomaly detection. It begins with feeding the captured network data (trace files), which is processed for data extraction according to a predefined feature set (FS). This extracted data are encoded with all the encoding schemes available within the framework to produce uniquely encoded data instances. For each encoding scheme, the corresponding transformed data are then utilized to build a separate trained instance of both *Autoencoder* models introduced in Section IV-B. Figure 1 exemplifies the ML pipeline. Similarly, each attack dataset is preprocessed and encoded to be evaluated with the classifier trained on the respective encoding scheme.

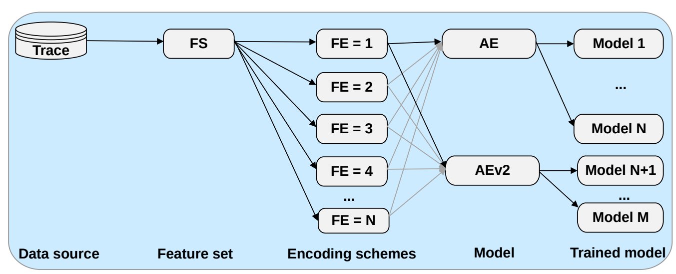

Fig. 1. ML Training Pipeline

A. Data Sources

The data were collected for internal use by capturing Ethernet network traffic from a real test vehicle to represent normal traffic behavior using appropriate tools such as *Wireshark*[3]. In order to replicate the malicious network activities, a set of network scanning routines were purposely carried out. The scans were executed from an additional device with physical access to the vehicle ethernet network. These network scans are typically performed by a threat actor in the initial phase of an attack chain, i.e., reconnaissance phase, to gather information about their target and identify

[3]https://www.wireshark.org/

vulnerabilities. Detecting such behaviors would serve as early indicators for potential attacks, and this could enhance other systems, such as SIEM (Security Information and Event Management), with additional data points and information to build a better situational awareness to make more suitable decisions for countermeasures.
Table II presents basic information about the captured data used as datasets for training and testing the classifier(s). The *noattack* dataset represents the typical intra-vehicular network communication. It is used for training the ML algorithm, and the remaining three datasets, designated as *attack* datasets, were used to assess the trained classifier for its detection performance. The anomaly percentage portrays the share of the attack data points in each individual dataset. It is worth pointing out that these datasets are proprietary and collected from a real preseries vehicle and are, therefore, not provided for open-source access due to its sensitive nature and intellectual property rights. The same holds true for the models' architectures, which has been sparsely introduced in Section IV-B.

TABLE II
BASIC INFORMATION ABOUT DATASETS

Data	**Packet Count**	**Anomaly Percentage**
noattack	603869	0.0
ArpScan	607850	0.3
HostScan	606533	0.3
PingScan	607662	0.5

As the data are available in the form of trace files, the feature extraction has been performed to derive the features from each packet in a packet-based approach, which goes into building the ML pipeline for training and testing activities.
The features extracted from the data sources are listed in Table III. Here, the feature *protocols* includes all protocols identified in the data, and the feature list is appended with a separate column for each protocol, and consequently 0s and 1s are placed respectively as their values, depending on their availability in the extracted data instance.

TABLE III
FEATURE LIST EXTRACTED FROM THE DATA

Index	**Feature Description**
0	*Source MAC Address*
1	*Destination MAC Address*
2	*Source IP Address*
3	*Destination IP Address*
4	*Source Port Number*
5	*Destination Port Number*
6	*Virtual LAN*
7	*Frame size*
8	*UDP payload size*
9	*Protocols*

B. Models

As the scope of the evaluation has been limited to unsupervised classification of anomalies in the network data, the state-of-the-art deep learning model *Autoencoder*, apparent from recent publications [12] and [13], has been chosen as classification algorithm for our empirical evaluation. Here, the two variants of deep learning *Autoencoder* models, namely sequential (*AE*) and functional (*AEv2*), have been implemented using Keras layers[4], and TensorFlow APIs[5]. The *Autoencoder* built using sequential APIs is a simpler architecture with layer-by-layer modelling, and is referred to as *AE* while the model *AEv2* is implemented using functional APIs, which offer greater flexibility to build complex architectures and non-linear connections. The inclusion of these models would enable the assessment of how the encoding schemes perform differently, and compare them against each other.

C. Thresholding mechanism

To accomplish the threshold determination, a python library named *PyThresh*[6] has been employed. *PyThresh* offers a range of thresholding schemes which rely on statistics of the likelihood or decision scores generated through a model for data instances. Among various schemes to establish threshold values, Median Absolute Deviation (MAD) was being used for threshold determination for the models in our framework. As MAD acts as a non-parametric means to estimate the threshold from the generated decision scores, the anomaly label is set to any decision score value reaching beyond the mean plus the median absolute deviation over the standard deviation.

Hence, the model's evaluation is initially executed on the training data once the model is fitted, which results in assigning the decision scores to all the data points. MAD function is then initialized and all the decision scores, once normalized, are brought in to evaluate a new threshold using (10) and (11), which is then used as a threshold to convert the scores into the predicted labels during the model evaluation.

$$MAD = med|x - med(x)|. \tag{10}$$

$$\text{threshold} = \bar{x} + \frac{MAD}{\sigma} \tag{11}$$

V. PERFORMANCE METRICS

A performance metric is aimed at indicating how good a trained model performs at a given classification task. This requires availability of the ground truth known as *Class* or *Label*. In our case, the MAC address of the network interface from which scans were performed, was leveraged for labelling each data point, either as '0' for denoting *noattack* or '1' for marking an *attack* association.

[4] https://github.com/fchollet/keras
[5] https://www.tensorflow.org/
[6] https://github.com/KulikDM/pythresh

Based on the ground truth, the Confusion Matrix (CM), consisting of four parameters such as True Negatives (TN), False Negatives (FN), False Positives (FP) and True Positives (TP), could be an intuitive way to measure performance for binary classification by any ML algorithm. Further parameters can be derived, such as accuracy, precision, recall, or specificity [14]. Precision and recall can be calculated as shown in (12) and (13) respectively. The harmonic mean of precision and recall, known as F1-score, could also be used for representation of the overall classification performance, as presented by (14). The value of F1-score ranges from [0,1] and its higher values indicate better classification performance.

$$Precision = \frac{TP}{TP\ +\ FP} \tag{12}$$

$$Recall = \frac{TP}{TP\ +\ FN} \tag{13}$$

$$F1\text{-}score = \frac{2\ *(precision\ *\ recall)}{(precision\ +\ recall)} \tag{14}$$

In this assessment, the classifier's detection performance is illustrated using the widely recognized metric, F1-score, which takes both the correct and missed detections into the account, as gathered from these above equations.

VI. Experimental Evaluation

A. Test Environment

A virtual Ubuntu 22.04.4 LTS server, equipped with 16 cores, 32 GB memory, and 200 GB disk, was set up as a testing environment for benchmarking our experiments. A virtual environment was created with Python 3.10.12 to have all the required packages installed for executing python scripts.

B. Evaluations based on Feature Encodings

As depicted in Figure 1 exemplifying the ML pipeline, the *noattack* dataset was preprocessed into a tabular data format and then encoded with each encoding technique to train a separate instance of the model(s). In each instance of the model training, the MAD function is employed to determine the threshold, as explained in Section IV-C. Similarly, each attack dataset is preprocessed and encoded to be tested with the classifier trained on the respective encoding scheme in the evaluation phase. As *IPv4* is still the dominant internet protocol being used in network infrastructures, this study has been kept limited to *IPv4* network data.

Tables IV and V shows the classification performance for each *Autoencoder* model separately for the encoding schemes against all three attack scenarios mentioned in Section IV-A. The encoding scheme(s) achieving the maximum F1-score for each individual attack scenarios is marked in bold.

TABLE IV
AE Performance in terms of F1-score

Number	**Encoding scheme**	**Attack Scenario**		
		ArpScan	***HostScan***	***PingScan***
0	IE	0.632	**0.535**	0.721
1	HE	0.0	0.0	0.0
2	OS	**0.784**	0.034	**0.839**
3	HSF	0.0	0.0	0.0
4	ISF	0.632	**0.535**	0.721
5	HVD	0.0	0.0	0.0
6	IVD	0.744	0.331	0.812
7	LVD	**0.784**	0.001	0.838
8	NIHI	0.782	0.001	0.837
9	NHHH	**0.784**	0.001	0.838

TABLE V
AEv2 Performance in terms of F1-Score

Number	**Encoding scheme**	**Attack Scenario**		
		ArpScan	***HostScan***	***PingScan***
0	IE	0.713	0.688	0.788
1	HE	0.0	0.0	0.0
2	OS	0.02	**0.836**	0.066
3	HSF	0.0	0.0	0.0
4	ISF	0.713	0.688	0.788
5	HVD	0.0	0.0	0.0
6	IVD	**0.845**	0.001	**0.884**
7	LVD	0.824	0.433	0.87
8	NIHI	0.822	0.416	0.863
9	NHHH	0.005	0.182	0.017

VII. Results and Discussion

A comparative assessment could be drawn up for the classification performance for the models trained with each encoded dataset on these three attack scenarios based on Figures 2 and 3. The numbers on the horizontal axis refer to the encoding schemes, as shown in Tables IV and V. It is evident that none of the encoding schemes could enable either of these *Autoencoder* models to achieve the best classification performance in all three attack scenarios simultaneously. While the *OS* encoding showed better performance with *AE* in classifying the attack scenarios like *ArpScan*, and *PingScan*, it, however, performed badly in identifying the *HostScan* attack data. In contrast, the model *AEv2* stood out in identifying only this *HostScan* attack scenario with the *OS* encoding. Similarly, with *IVD* encoding scheme, the model *AEv2* apparently showed quite good performance in analyzing the *ArpScan* and *PingScan* datasets, but performed poorly with the *HostScan* datasets. However, it could be learnt that the encoding schemes like *IE*, *ISF*, and *IVD* achieved a reasonable performance with the *AE* model in all three attack scenarios. Similarly, the model *AEv2* showed reasonably good performance in all three scenarios with the encoding schemes like *IE*, *ISF*, *LVD*, and *NIHI*. Furthermore, it could also be ascertained that all hash-based encodings did not perform well

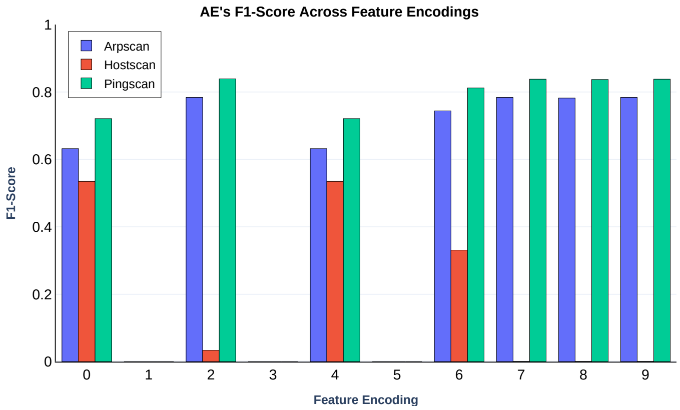

Fig. 2. AE Classification Performance

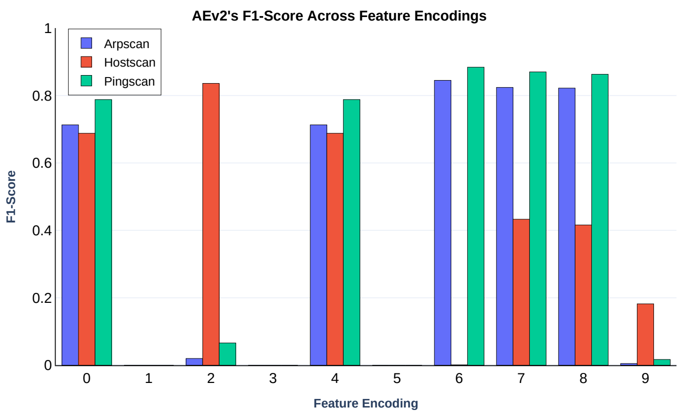

Fig. 3. AEv2 Classification Performance

This outcome for hash encoding could be attributed to loss of semantic proximity wherein, unlike other encodings, all the categorical values are treated randomly and independently. When encoded with hash-based encodings, unrelated features may be assigned numerically close values, leading the model to infer a false sense of similarity. Alternatively, two similar values may be given distant values, causing them to be treated as differing feature values. This phenomenon obscures the relationship between feature values that the models rely on to assimilate ingrained patterns in the data.

VIII. Conclusion

This paper presented the feature encoding schemes for transforming categorical features in network data for handling ML classification tasks, limiting the need to raise the data cardinality to only a certain degree. These feature encoding schemes were used to transform the network traffic data captured from a real vehicle, and the encoded data were used to build an ML pipeline for the classification task. Two variants of *Autoencoder* models were trained with training data encoded with each encoding scheme, and were evaluated on three respective encoded attack datasets for examining their classification performance.

The results showed that the feature encoding certainly has an impact on the classification performance of the models used, which can be noted from their respective performance tables as well as from the plots. Furthermore, it can also be observed that the model *AEv2* achieved higher performance in classifying all three attack scenarios when compared with other *AE* model, not only individually but also simultaneously with some specific encoding scheme such as *IE* and *ISF*.

In our future work, an exhaustive study is aimed to be performed to explore ways to improve the encodings further to achieve a higher classification performance across various attack scenarios. In addition, Explainable AI *(XAI)* aspects are to be explored to gain deeper understanding on how the encodings impact the classification predictions, which could be considered vital in respect to the downstream processing of these outcomes such as signature generation and attribution [15].

References

[1] K. Matheus and T. Königseder, *Automotive ethernet*. Cambridge University Press, 2021.

[2] S. Koduru, S. P. Machina, S. Madichetty, and S. Mishra, "Data-driven solutions for next-generation automotive cybersecurity," *Transactions of the Indian National Academy of Engineering*, pp. 1–20, 2024.

[3] A. Zheng and A. Casari, *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[4] D. Breskuvienė and G. Dzemyda, "Categorical feature encoding techniques for improved classifier performance when dealing with imbalanced data of fraudulent transactions," *International Journal of Computers Communications & Control*, vol. 18, no. 3, 2023.

[5] E. Shao, "Encoding IP Address as a Feature for Network Intrusion Detection," thesis, Purdue University Graduate School, Dec. 2019.

[6] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.

[9] J. M. Johnson and T. M. Khoshgoftaar, "Encoding techniques for high-cardinality features and ensemble learners," in *2021 IEEE 22nd international conference on information reuse and integration for data science (IRI)*. IEEE, 2021, pp. 355–361.

[10] M. Ring, A. Dallmann, D. Landes, and A. Hotho, "Ip2vec: Learning similarities between ip addresses," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2017, pp. 657–666.

[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[12] F. Harrou, B. Bouyeddou, A. Dairi, and Y. Sun, "Exploiting autoencoder-based anomaly detection to enhance cybersecurity in power grids," *Future Internet*, vol. 16, no. 6, p. 184, 2024.

[13] R. Singh, N. Srivastava, and A. Kumar, "Network anomaly detection using autoencoder on various datasets: A comprehensive review," pp. 63–77, 2024.

[14] M. Heigl, K. A. Anand, A. Urmann, D. Fiala, M. Schramm, and R. Hable, "On the improvement of the isolation forest algorithm for outlier detection with streaming data," *Electronics*, vol. 10, no. 13, p. 1534, 2021.

[15] M. Heigl, E. Weigelt, A. Urmann, D. Fiala, and M. Schramm, "Exploiting the outcome of outlier detection for novel attack pattern recognition on streaming data," *Electronics*, vol. 10, no. 17, p. 2160, 2021.