DTCEncoder: A Swiss Army Knife Architecture for DTC Exploration, Prediction, Search and Model Interpretation

Author names hidden
for the Review
and Submission

Abstract—**Diagnostic Trouble Code (DTC) events, produced in vehicles, assist in knowing the occurrence of faults in different modules and can be used for predictive maintenance by detecting patterns. While performing Exploratory Data Analysis (EDA) or correlating specific DTC events is an easy task, searching for patterns in long multivariate DTC sequences can be very challenging. Instead of performing analysis for individual DTCs, a self-supervised approach using a Long Short-Term Memory (LSTM) network was introduced recently to perform the next DTC prediction. Despite its merits, such an approach is not interpretable for engineers who need to understand the decision-making process of the model. In this paper, we introduce the DTCEncoder, a recurrent neural network that incorporates Gated Recurrent Units (GRU) and an attention mechanism to encode DTC sequences into low dimensional representations, and that serves as a unified approach to (i) efficiently represent multivariate event sequences and predict the next event, (ii) interpret what the model learns and what it uses for the next prediction, and (iii) perform efficient semantic search for individual DTCs and DTC sequences.**

Index Terms—**Predictive maintenance, DTCs, RNN, Encoder, Interpretibiltiy.**

I. Introduction

To prevent the unexpected failure of industrial machines, periodic and scheduled maintenance is usually performed to take proactive measures. This periodic approach commonly named *Preventive maintenance*, is being replaced by a cost-effective and data-driven failure prediction approach known as *Predictive Maintenance*, which helps to foresee the necessity of maintenance based on data collected from sensors.

Modern vehicles are also equipped with diagnostic modules that produce DTCs, which can, in turn, be analyzed to find patterns for predictive maintenance. These fault events are characterised by multiple attributes corresponding to different granularity levels and the details of previous faults. For instance, a single DTC event in a sequence of DTCs contains information about the main *module* where a fault has occurred, a *sub-module* (often called base-dtc), and a *fault-type*, which specifies the exact nature and granularity of the fault.

In comparison to sensory data, modeling DTCs is a relatively difficult task as they work on non-numeric attributes with a high number of unique categories (cardinality). As a result, most of the research in the area has focused on developing simple models that only consider individual or small groups of DTCs and that do not cater to complex sequential dependencies of sequences. Some models have introduced learning mechanisms to address such issues, yet they are not interpretable. For example, [1] used EDA to select features to train a machine learning algorithm (AdaBoost) to detect failures associated with a single module, namely a power motor, and classification approaches (Decision Trees, Support Vector Machines, and Random Forest) have been employed to distinguish faulty from non-faulty patterns as well as to search for fault patterns, in datasets containing limited groups of DTCs (e.g., [2] [3] [4]).

Deep learning models that consider sequential dependencies and solve the above-mentioned representation issues have also been proposed: in [5], a convolutional neural network and a gated recurrent network were used to learn embeddings for specific errors and their relationships; more in particular, the LSTM-based Sequential Multivariate Fault Prediction (SMFP) model [6] shows a promising direction for fault prediction [1] using a self-supervised end-to-end architecture. This task is illustrated in figure 1, where instead of providing the labels explicitly, the last event of each sequence is treated as the next target label. The predicted DTC event includes three attributes of different granularity (i.e., fault-type) and can be utilized to perform primitive measures on the predicted module.

Unlike previous DTC-based approaches, SMFP does not limit modelling to few DTCs but rather it uses neural embeddings [8] to represent non-numeric attributes and handles sequential dependencies with recurrent neural networks. However, it lacks interpretability, which is often required to ascertain what the model considers before predicting the next item. To interpret, verify and compare the prediction of multivariate fault events, an engineer will need to perform a tedious, and often impossible, task of searching millions of similar multivariate sequences. Engineers are thus forced to develop different retrieval, modeling, and exploratory techniques for sequential prediction models, which do not provide a holistic view.

[1] [5] also performs a *sample* task of *part failure predictions* to evaluate static CBOW embeddings [7]. However, these embeddings are not optimized with the downstream *part failure predictions*, but rather they are learned separately and used as a pre-trained layer in such sample task.

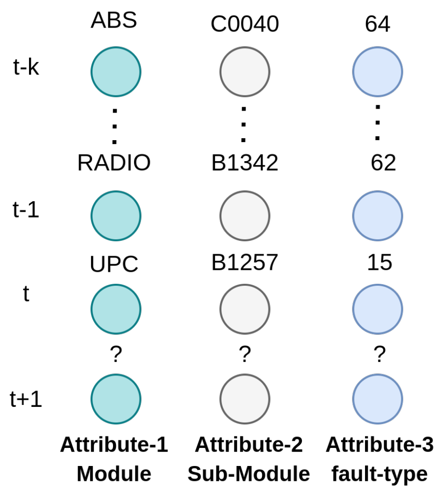


Fig. 1. Single sequence showing multivariate DTC events from time t_k to time t. The goal is to predict the multivariate event at time $t+1$.

In this paper we propose an attention-based DTCEncoder, which aims to learn low compact representations of multivariate event sequences and hence enable interpretable multivariate DTC event prediction. With the use of the Luong attention [9] mechanism, these representations explicitly incorporate information from all previous hidden states according to their importance, which procures interpretability and also improves prediction accuracy. The DTCEncoder provides a unified approach to predict the next DTC event, efficiently represents a pipeline for model interpretation, reduces search space for the semantic search of multivariate sequences, and performs event and sequence level EDA.

In the next section, we comment on related work and on the background of the proposed model. In section III, we present the methodology used by DTCEncoder for prediction, interpretation, and efficient sequence representation learning. The experiments we carried out and their results are described in the next section. We shall finish with conclusions.

II. Background and Related work

In this section, we provide a brief review of the SMFP model. The sequential event prediction problem, specially in the context of DTC events, can be defined as predicting the DTC fault event at timestep $t+1$ given a sequence of DTCs from timestep 1 to the current timestep t. To predict the next event, the model needs to account for sequential dependencies in multivariate faults. A single DTC event comprises of three different attributes (variables), which provide different levels of information about the fault. A vehicle can contain several Electronic Control Unit (ECU) resulting in different DTCs. The first attribute (module) of a DTC signifies which ECU has generated the DTC fault. The second attribute (sub-module) corresponds to a so called base-DTC and the third attribute (fault-type) provides the information about the type of the fault that has occurred. For example, in figure 1 we depict a sequence $S =$ [(ABS,C0040,64),....,(RADIO,B1342,62),(UPC,B1257,15)] with T DTC events, each consisting of three attributes , namely, (i) a module (e.g., ABS), (ii) a sub-module or base-DTC (e.g., C0040) and (iii) fault-type (e.g., 62). The goal is to predict an event at timestep $t+1$ with all the three attributes.

Formally, let sequence $S = e_{1:T}$ of DTC events, containing T observed events, where each event e_t at timestep t represents one DTC event of the form (a_t^1, a_t^2, a_t^3). Here, a_t^i represents the *i-th* attribute (feature) of the event, for example, a main module where a fault is detected. If θ denotes the parameters of a model that we want to learn, then given a sequence of T observed events, the event $e \in E$ with the highest probability will be predicted as the next event, following

$$\underset{i}{\operatorname{argmax}}\, P(e_i | e_{1:T}, \theta) \tag{1}$$

Since the cardinalities of attributes are high, separate neural embedding layers were used to learn dense representations a^i_{EMB} for each attribute a^i, such that $size(a^i_{EMB}) < size(a^i)$. Embedding layers project the One-Hot Encoded (OHE) representation of textual attributes to a continuous representation, which is jointly optimized with the downstream prediction task. SMFP concatenates individual attribute embeddings to form an embedding matrix A_{EMB} and passes it to LSTM layers for identification of sequential dependencies. Finally, three dense output layers were employed to predict individual attributes of the events.

III. Methodology

In this section, we will first briefly introduce the overall architecture of the DTCEncoder, followed by details of different components and downstream tasks.

A. *DTCEncoder motivation*

In SMFP [6], the authors showed how LSTMs can predict the next DTC. A LSTM is a Recurrent Neural Network (RNN) where a recurrent loop acts as a memory to handle sequential dependencies. It overcomes the vanishing and exploding gradient problems in RNNs, by introducing gates that channel access to the long-term memory.

The representations learned by LSTMs are typically high dimensional, for example, 128 to 512 dimensional vectors. It is hard to interpret what an LSTM has learned with these representations. Although different unsupervised learning techniques like dimensionality reduction [10] can be used for the sake of interpretation, reducing a large dimensionality space usually causes loss in information, specially about the sequential nature of data. By applying the attention mechanism on hidden states of a special type of RNN, namely Gated Recurrent Units (GRU) and forcing the information to pass through a dense bottleneck, the DTCEncoder learns a compact low dimensional DTC representation (context-vector), which not only improves the accuracy of the DTC prediction task but also provides interpretation and efficient semantic search

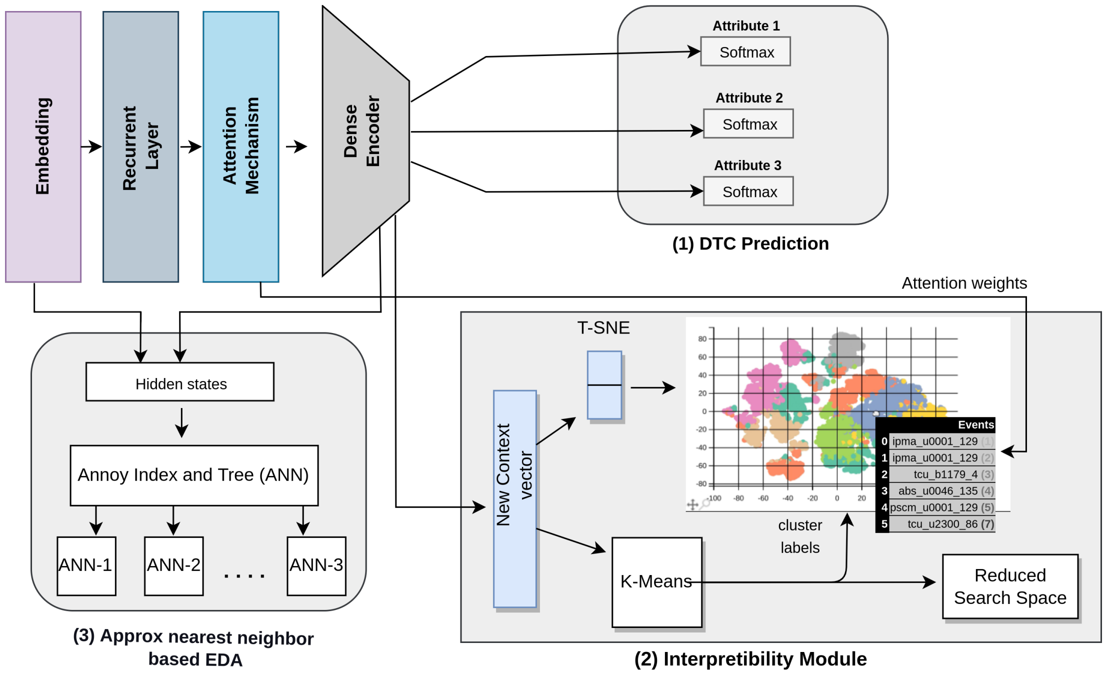


Fig. 2. The overall architecture of the DTCEncoder. Component-1 shows the encoding unit, which learns a low dimensional representation for each DTC sequence with the help of the attention mechanism, the GRU, and the dense encoder. The interpretability module (Component-2) utilizes context-vectors and attention-weights produced by component-1, to perform dimensionality reduction and visualization. Component-3 corresponds to ANN-based semantic search unit, which enables EDA and fast retrieval of individual DTCs and sequences using hidden representation learned by component (1).

of DTC sequences (and individual DTCs). The overall architecture of the DTCEncoder can be seen in figure 2 and the procedure is explained in Algorithm 1.

We are providing the specifics of the encoding mechanism in the next sub-section.

Algorithm 1 DTCEncoder

```
Require: $s_1 \dots s_N$
Ensure: Predicted DTC event ($a_p^1$,$a_p^2$,$a_p^3$) for $s_1 \dots s_N$
  for $epoch \leftarrow 1$ to $N$ do
    $a_{EMB}^1 \leftarrow EMB(a_{OHE}^1)$
    $a_{EMB}^2 \leftarrow EMB(a_{OHE}^2)$
    $a_{EMB}^3 \leftarrow EMB(a_{OHE}^3)$
    $a_{EMB} \leftarrow a_{EMB}^1 \parallel a_{EMB}^2 \parallel a_{EMB}^3$ {Concatenating the individual embeddings}
    $gru\_state \leftarrow GRU_1(a_{EMB})$
    $gru\_state \leftarrow GRU_2(gru\_state)$
    $lastSt$ , $stWithoutLast \leftarrow slice(gru\_state)$
    $Score$,$Context \leftarrow Attention(lastSt, stWithoutLast)$
    $Dense1 \leftarrow Dense(Context)$
    $DenseBottleneck \leftarrow Dense(Dense1)$
    $a_{pred}^1 \leftarrow Dense(DenseBottleneck)$
    $a_{pred}^2 \leftarrow Dense(DenseBottleneck)$
    $a_{pred}^3 \leftarrow Dense(DenseBottleneck)$
    Calculate loss
    Optimize parameters of all layers
  end for
```

B. Attention mechanism and Encoder

The LSTM architecture used in SMFP [6] takes concatenated attribute embeddings A_{EMB} and passes them through encoder LSTM layers. LSTM layers first calculate the new hidden state from the current input x_t and the previous hidden state h_{t-1}. The prediction is then calculated by utilizing information learned by the hidden state h_t, up to the current timestep t.

Instead of predicting the next DTC based on the last hidden state (context-vector), we use a Loung attention mechanism [9] to obtain a new context-vector by taking a weighted sum of the hidden states from all time steps, where the weights are calculated with an attention mechanism.

We use the last T DTC events as training inputs and keep the event at timestep $(T + 1)$ as the target. The goal of the model is to predict this event as the next DTC. Moreover, we append an end-of-sequence (EOS) token to all input sequences. As figure 3 depicts, to obtain a new weighted context-vector, we first take a dot product $e_t = h_{EOS} \cdot h_t$ between each hidden state h_t and the hidden state of the EOS token. We then pass these scores to a softmax function

$$\alpha_{t,j} = \frac{exp(e_{t,j})}{\sum_{i=1}^{T} exp(e_{t,i})}, \tag{2}$$

where $\alpha_{t,j}$ is the attention weight at timestep t for the hidden-state j, and $e_{t,j}$ is the attention-score (before applying softmax) for the same timestep.

Softmax is used to obtain attention weights for all hidden-states (h_t) by normalizing the attention score. We calculate a

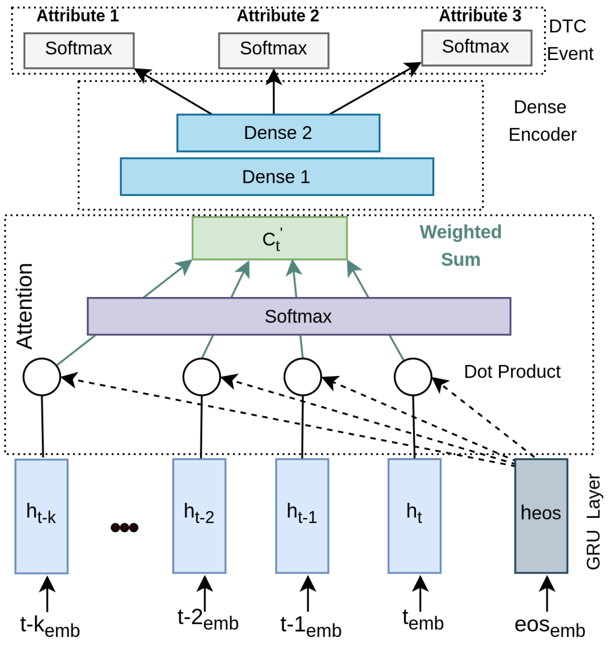


Fig. 3. The attention mechanism performs a dot product operation between all hidden-states h_t and the EOS hidden-state. The new context-vector $\hat{c}_t$ is obtained by performing the weighted sum of all hidden states. The new context-vector is passed through dense layers with a decreasing number of neurons (Encoder), and finally to the output layers of all three attributes.

weighted context-vector ($\hat{c}_t$) according to the attention weights (α) produced by softmax, as shown in the equation below

$$\hat{c}_t = \sum_{i=1}^{T} \alpha_{(t,i)} \cdot h_i \qquad (3)$$

The new context-vector is passed through dense layers with a decreasing number of neurons (encoder (E)) to obtain a compact encoding for the sequence. The output of final hidden dense layer (bottleneck) is then passed to the three dense output layers, each corresponding to a different attribute of the DTC event.

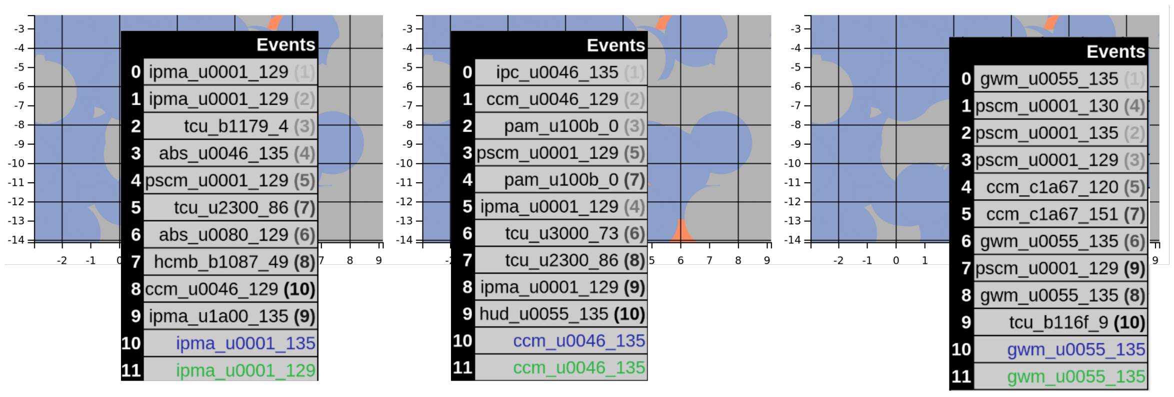
C. Clustering and dimensionality reduction for the interpretable visualization module

The sequence representation (context-vector) learned by the DTCEncoder is a 24-dimensional vector and much smaller than the representation learned by a typical RNN network, which usually is at least a 128 to 512-dimensional vector. Performing dimensionality reduction from a hidden state of the RNN layer is difficult, as compared to reducing a 24-dimensional vector. We apply the dimensionality reduction technique t-SNE [11] to this 24-dimensional representation to further reduce it to 2 dimensions, which can thus be visualized.

As seen in component (2) of figure 2, the interactive module of the DTCEncoder applies the K-means clustering algorithm to the event-context (E_C) of all sequences and clusters them into K groups. These clusters provide two benefits: first, they help to visualize the clusters in a 2-dimensional space by enabling the same color-coding of sequences within a cluster; second, when performing a semantic search for a long multi-variate DTC event sequence, they provide an added performance verification metric for the retrieval mechanism and can also be used to improve retrieval itself.

As shown in figure 4, we project the sequence onto 2-dimensions learned with t-SNE and assign them colors according to their K-means clusters. We visualize them such that upon moving the cursor onto each sequence, we can see the N previous timesteps, the actual next timestep event that the model needed to predict (in blue), and the prediction made by the model (in green). The clustering and dimensionality reduction procedure is defined in Algorithm 2.

Since the representation learned by RNNs regards all sequential information, the close sequences in this 2-dimensional space should be the ones that lead to the same fault or have the same sequential context. This hypothesis is confirmed by zooming in on this interactive visualization, which shows that the sequences in the close neighborhood are similar or have the same DTC at the last timestep. The attention weights associated with the hidden state of each timestep show how important that particular timestep is for the model to predict the next DTC. We use these attention weights to enhance the interpretability module by highlighting DTC events in a way that reflects their importance. As seen in the figure 4, the events that have higher attention weights score a higher importance rank (10 being the most important), with higher opacity than the other events. This interactive module provides a very efficient way of seeing what the model has learned and which contextual similarities are present in the dataset.



Events	
0	ipma_u0001_129 (1)
1	ipma_u0001_129 (2)
2	tcu_b1179_4 (3)
3	abs_u0046_135 (4)
4	pscm_u0001_129 (5)
5	tcu_u2300_86 (7)
6	abs_u0080_129 (6)
7	hcmb_b1087_49 **(8)**
8	ccm_u0046_129 **(10)**
9	ipma_u1a00_135 **(9)**
10	ipma_u0001_135
11	ipma_u0001_129

Events	
0	ipc_u0046_135 (1)
1	ccm_u0046_129 (2)
2	pam_u100b_0 (3)
3	pscm_u0001_129 (5)
4	pam_u100b_0 (7)
5	ipma_u0001_129 (4)
6	tcu_u3000_73 (6)
7	tcu_u2300_86 **(8)**
8	ipma_u0001_129 **(9)**
9	hud_u0055_135 **(10)**
10	ccm_u0046_135
11	ccm_u0046_135

Events	
0	gwm_u0055_135 (1)
1	pscm_u0001_130 (4)
2	pscm_u0001_135 (2)
3	pscm_u0001_129 (3)
4	ccm_c1a67_120 (5)
5	ccm_c1a67_151 (7)
6	gwm_u0055_135 (6)
7	pscm_u0001_129 **(9)**
8	gwm_u0055_135 **(8)**
9	tcu_b116f_9 **(10)**
10	gwm_u0055_135
11	gwm_u0055_135

Fig. 4. The interpretability module maps sequences to a 2-dimensional t-SNE space and color codes them according to the K-means clusters learned on their context-vector. This figure shows three different sequences in proximity by hovering different positions in the interpretability module. Each sequence presents all input events (black), the target event (blue), and the predicted event (green). The number in front of input DTC events is an attention score, which signifies the importance of the event between 1 (least important) and 10 (very crucial). It can also be seen that items near in the visualization have common DTCs. For example, a few modules (IPMA, TCU, CCM, etc.) and sub-modules (u0001, u0046, u0055, etc.) appear frequently in all three sequences.

Algorithm 2 Clustering and t-SNE

```
Require: Encoded States for $s_1 \dots s_N$
Ensure: Clusters and Reduced dimension $R^2$
    K-means ← fit K-means to encoded states of all $S$
    retrieve cluster from K-means
    t-SNE ← fit t-SNE to encoded states of all $S$
    Retrieve $R^2$ representation for each $S$ from t-SNE
```

D. Exploratory analysis and semantic search of DTCs and DTC sequences

The interactive module explained in the last section provides an efficient sequence visualization mechanism for explaining the behavior of the model. It can also be helpful for engineers to search for similar DTCs (or DTC sequences) to understand the semantics and context of the faults. Although possible, searching for the exact match within the whole sequence dataset or finding patterns can be computationally expensive, especially with the increase of data points and number of attributes.

Instead of performing an exact search for sequential or individual DTCs, we propose to employ an Approximate Nearest Neighbors (ANN) search on representations learned by two components, namely, (i) a concatenated attribute embedding for individual DTCs, and (ii) a bottleneck encoder representation of the DTCEncoder (context-vector) for sequences. Searching DTCs with the ANN Index reduces retrieval time and provides semantic information to find patterns in DTC sequences. Component (3) of figure 2 provides a visual flow of the Indexing and exploratory analysis pipeline.

We build an ANN Index with an ANN algorithm called *Annoy* [12]. Annoy uses a random projection to build a binary tree, which partitions the space such that it keeps similar points close in the tree representation. We build an Annoy Index to learn $\log(N)$ trees from the vectors of all N entities (sequences or individual DTCs) and later use this Index to search K nearest neighbors for a DTC or a given sequence.

TABLE I
TOP 3 NEIGHBORS OF FREQUENT OCCURRING DTC FAULTS

ANN	IPC_U0055_135	CHCM_U0046_129	ABS_U0046_135
1	IPC_U0055_130	CHCM_U0046_135	ABS_U0046_129
2	GWM_U0055_135	CHCM_U0046_130	ABS_U0046_130
3	IPC_U0001_135	PAM_U0046_129	ABS_U0046_136

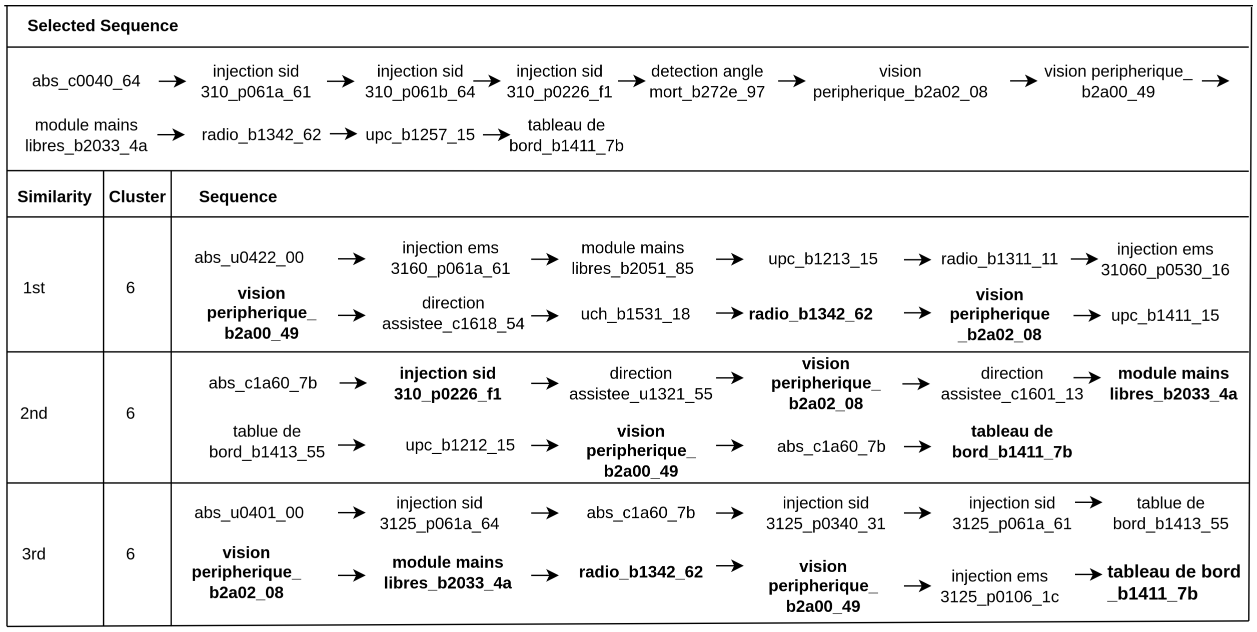
The procedure for semantic search is defined in Algorithm 3.

For individual DTCs, we retrieve the concatenated attribute embedding learned by the embedding layer and apply an Annoy Index on it. Since the embedding learned by the DTCEncoder is jointly optimized with the task in hand, i.e, the next fault prediction, the nearest neighbors of the faults are the ones that appear in the same context. Table I shows three nearest neighbors to a given DTC, comprising of three attributes (module_sub-module_fault-type). For example, in the third column, all three nearest neighbors share the same module attribute (ABS) and the same sub-module (U0046).

Algorithm 3 ANN with Annoy Index

```
Require: All DTC sequences ($s_1 \dots s_N$)
Ensure: Annoy Index for DTCs
1: Retrieve context-vector or concatenated attribute embedding for all unique DTC sequences (or faults)
2: Index ← build Index for all Sequences (or DTC events)
3: Return Index
```

For sequences, we learn an Annoy Index on the 24-dimensional bottleneck representation (context-vector) of all the DTC sequences. At the time of query, we retrieve a representation of the query sequence by passing it to the encoder and perform an ANN search for such sequence. As seen in figure 5, the most similar sequences retrieved by the Index are the ones that have more DTCs in common or end in the same faults (DTCs).



Selected Sequence

abs_c0040_64 → injection sid 310_p061a_61 → injection sid 310_p061b_64 → injection sid 310_p0226_f1 → detection angle mort_b272e_97 → vision peripherique_b2a02_08 → vision peripherique_ b2a00_49 → module mains libres_b2033_4a → radio_b1342_62 → upc_b1257_15 → tableau de bord_b1411_7b

Similarity	**Cluster**	**Sequence**
1st	6	abs_u0422_00 → injection ems 3160_p061a_61 → module mains libres_b2051_85 → upc_b1213_15 → radio_b1311_11 → injection ems 31060_p0530_16 → **vision peripherique_ b2a00_49** → direction assistee_c1618_54 → uch_b1531_18 → **radio_b1342_62** → **vision peripherique _b2a02_08** → upc_b1411_15
2nd	6	abs_c1a60_7b → **injection sid 310_p0226_f1** → direction assistee_u1321_55 → **vision peripherique_ b2a02_08** → direction assistee_c1601_13 → **module mains libres_b2033_4a** → tablue de bord_b1413_55 → upc_b1212_15 → **vision peripherique_ b2a00_49** → abs_c1a60_7b → **tableau de bord_b1411_7b**
3rd	6	abs_u0401_00 → injection sid 3125_p061a_64 → abs_c1a60_7b → injection sid 3125_p0340_31 → injection sid 3125_p061a_61 → tablue de bord_b1413_55 → **vision peripherique_ b2a02_08** → **module mains libres_b2033_4a** → **radio_b1342_62** → **vision peripherique_ b2a00_49** → injection ems 3125_p0106_1c → **tableau de bord _b1411_7b**

Fig. 5. Example of three similar sequences provided by ANN, for a selected DTC. DTCs which are common in the selected DTC and similar DTCs are highlighted in bold. For example, the selected sequence, the second, and the third similar sequence all contain specific DTCs (e.g., vision peripherique_b2a02_08) and end in the same DTC event (tableau de bord_b1411_7b)

IV. Experiments and Results

In this section, we are first sharing details about the dataset. Next, we explain the experimental setup and hyperparameter choices. Finally, we present the results of different experiments.

A. *Dataset*

We used the same dataset from [3] to train the DTCEncoder. This dataset is given by *name hidden for the review and submission*. As defined in section I and depicted in figure 1, each sequence contains multivariate fault events and corresponds to unique vehicles. Out of 2,50,000 fault sequences, we used 232,750 sequences for training, 12,500 separate sequences for testing, and 4,750 different sequences for validation.

B. *Experimental setup and hyperparameter tuning*

The hyperparameters and model parameters defined in this section are opted with the Hyperband [13] hyperparameter tuning method using keras-tuner [14]. The attribute embedding sizes that we considered, along with the sizes selected by tuner, are mentioned in table II. We took the last 10 timesteps of each sequence and padded the shorter sequences with 0, which were later masked (ignored) by subsequent layers. Apart from the embedding size, other hyperparameter choices (e.g., number of GRU units in each GRU layer) are listed in table II. We used Adam optimizer [15] with a learning rate of 0.0066.

We used two GRU layers with 128 GRU cells each. To improve generalization, we used a recurrent-dropout of 0.3 and a dropout of 0.2 in encoder layers. For dense output layers, we used a softmax as an activation function. In GRU layers, we used $tanh$ as the activation function. We used categorical cross entropy loss for all outputs.

TABLE II
Hyper-parameter and parameter choices

Choice	min	max	final
Attribute-1 (module) embedding	4	24	6
Attribute-2 (base-dtc) embedding	4	32	12
Attribute-2 (fault-byte) embedding	12	56	8
GRU layers recurrent-dropout	0.1	0.5	0.3
GRU layers dropout	0.1	0.5	0.2
GRU layer units	128	256	224
Dense first layer units	32	128	64
Learning rate	1e-4	0.1	0.006

C. *Results*

TABLE III
Ablation study and results of DTCEncoder in comparison with the SMFP's.

Architecture	Validation Loss	Top-5 test accuracy
SMFP	4.50	76.15%
DTCEncoder (GRU)	4.36	79.21%
Without Attention and Encoder	4.49	76.0%
DTCEncoder (LSTM)	4.59	75.20%
DTCEncoder (RNN)	4.71	73.0%

The main focus of the proposed model was to provide an interpretable and unified architecture for multiple DTC-related tasks. Besides achieving the main objective of interpretability, the DTCEncoder also improved the previous baseline on the next event prediction task. We compared the performance of the DTCEncoder only against the LSTM architecture used in the SMFP approach [6], since, as far as we know, there is no other model that performs such task. We achieved the lowest validation loss of 4.36 and top-5 accuracy of 79% with our architecture, in contrast to validation loss of 4.52 and top-5 test accuracy of 76% with the architecture used by the SMFP model. The top-5 prediction accuracy of 79%, achieved for three combined high cardinality attributes, is different and more complex to attain than the accuracy metric used in a binary classification task.

As a part of the ablation study and to understand the importance of the DTCEncoder's core components, we removed the attention module and the dense layers preceding output layers. As shown in table III, the model was not able to meet the performance of the actual DTCEncoder architecture. We also used different recurrent networks like LSTM and simple RNN cells without gating, and we found that LSTMs started to overfit quite early and RNNs were not able to learn the patterns accurately.

The dataset used by the DTCEncoder has only 250,000 sequences and shows some limitations like missing events in a few sequences. So, we can expect improved performance with larger and better datasets.

V. Discussion and Conclusion

We have presented a unified architecture, the DTCEncoder, for multi-attribute sequential DTC event prediction, which interprets the model, learns to represent DTC sequences and can be used to perform semantic analysis at both sequence and individual DTC event levels. This model encodes a DTC sequence into a low-dimension representation, which encapsulates the sequential information that can be used for different downstream tasks. Along with enabling multiple functionalities, it surpasses the performance of SMFP [6].

Although some solutions for representing non-numeric (high cardinality) attributes and sequential dependencies have been proposed with neural embeddings and RNNs, such models still lack interpretability. The DTCEncoder makes the next DTC prediction mechanism interpretable, without the need for a separate procedure. The benefit that this unified architecture introduces is that the performance of the other components (e.g., interpretation, semantic search) will improve automatically with the increase of performance in the encoder component, and thus it does not require to maintain and retrain separate models for all downstream tasks. We are confident that with hyperparameter optimization and architectural adjustments, the performance of the DTCEncoder can be improved further.

References

[1] L. Virkkala and J. Haglund, "Modelling of patterns between operational data, diagnostic trouble codes and workshop history using big data and machine learning," Ph.D. dissertation, Uppsala universitet, 2016. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:909003/FULLTEXT01.pdf

[2] M. Fransson and L. Fåhraeus, "Finding patterns in vehicle diagnostic trouble codes : A data mining study applying associative classification," 2015. [Online]. Available: http://uu.diva-portal.org/smash/get/diva2:828052/FULLTEXT01.pdf

[3] U. Shafi, A. Safi, A. Shahid, S. Ziauddin, and M. Saleem, "Vehicle remote health monitoring and prognostic maintenance system," *Journal of Advanced Transportation*, vol. 2018, pp. 1–10, 01 2018.

[4] P. Pirasteh, S. Nowaczyk, S. Pashami, M. Löwenadler, K. Thunberg, H. Ydreskog, and P. Berck, "Interactive feature extraction for diagnostic trouble codes in predictive maintenance: A case study from automotive domain," in *Proceedings of the Workshop on Interactive Data Mining*, ser. WIDM'19. Association for Computing Machinery, 2019.

[5] K. R. Thoorpu and N. Prafulla, "Sequential dtc vector embedding using deep neural networks for industry 4.0," in *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*, 2020, pp. 912–915.

[6] A. Hafeez, E. Alonso, and A. Ter-Sarkisov, "Towards sequential multivariate fault prediction for vehicular predictive maintenance," in *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2021.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Proceedings of Workshop at ICLR*, vol. 2013, 01 2013.

[8] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, p. 1137–1155, 2003.

[9] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015. [Online]. Available: https://arxiv.org/abs/1508.04025

[10] L. van der Maaten, E. Postma, and H. Herik, "Dimensionality reduction: A comparative review," *Journal of Machine Learning Research - JMLR*, vol. 10, 01 2007.

[11] L. van der Maaten and G. Hinton, "Viualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[12] Spotify, "Annoy." [Online]. Available: https://github.com/spotify/annoy

[13] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018. [Online]. Available: http://jmlr.org/papers/v18/16-558.html

[14] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, "Kerastuner," https://github.com/keras-team/keras-tuner, 2019.

[15] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.