DTC-TranGru: Improving the performance of the next-DTC Prediction Model with Transformer and GRU

Authors removed for the double-blind review

ABSTRACT

Over the last few years, vehicular predictive maintenance has witnessed a shift from utilizing raw sensor reading directly to using fault events registered in On-Board Diagnostic systems (OBDs). Instead of providing raw sensory data, OBDs equip drivers and technicians with diagnostic information coming from different Electric Control Units (ECUs) in the vehicles, usually indicated as Diagnostic Trouble Codes (DTCs). These DTCs are categorical (non-numeric) or alphanumeric, and relate to different problems within the vehicle. Having many categories and multiple attributes has previously restricted researchers to analyzing a few DTCs at a time, with a limited set of machine learning algorithms. This has recently changed with the advent of the self-supervised next DTC approach, which ranges from an LSTM-based multivariate next-prediction model to an attention mechanism and transformer-decoder model. These models reframe the problem of predictive maintenance as the next fault event prediction task and use metrics like top-3 and top-5 accuracy to evaluate the predictive capabilities of the model. We propose a new architecture for the next DTC prediction task, DTC-TranGru, which combines the benefits of transformer and GRU models and shows that it outperforms them with around 2% increase in the top-5 accuracy benchmark for a next-DTC prediction task.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; **Learning latent representations**.

KEYWORDS

Predictive maintenance, Diagnostic Trouble Code, GRU, Transformers, Attention Mechanism, Embeddings.

ACM Reference Format:
. 2024. DTC-TranGru: Improving the performance of the next-DTC Prediction Model with Transformer and GRU: Authors removed for the double-blind review. In *Proceedings of ACM SAC Conference (SAC'24).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Predictive maintenance, which relies on using machine learning and data analytic methods to forecast the need for maintenance of industrial machines, has historically been performed using data recovered from sensors placed in these devices [6], [23]. Such sensory data is represented numerically and fed into anomaly detection algorithms such as clustering [24] and support vector [7] machines that extract useful patterns, hence predicting the need for the next maintenance episode. With the advent of automatic diagnostic systems, like On-Board Diagnostic Systems (OBDs) in vehicles, there has been a focus on analyzing and using Diagnostic Trouble Codes (DTCs) retrieved from them [17],[22], [10].

Notwithstanding their utility and pervasiveness in the automotive sector, the application of standard machine learning techniques for predictive maintenance using DTCs is not straightforward: DTCs are non-numeric, the cardinality of DTC codes is too high, and, in some cases, they coexist with other attributes like fault-byte and Electric Control Units (ECUs), which provide different information about the location and type of the fault. These restrictions have forced researchers to study a limited number of DTCs at a time and use simple algorithms such as [17], [22], [5]. The challenge that this paper addresses is the development of deep learning algorithms that work directly with the inherent complexity of DTCs.

There is a second caveat that needs to be tackled: Algorithms applied to DTCs mostly make use of repair and warranty data in order to cast the problem into a supervised learning approach, for example, the classification of faulty and non-faulty sequences of events. In the absence of such data and a lack of clues about when the vehicle comes to a standstill, it is difficult and often impossible to apply supervised learning techniques that require large samples of labeled data.

Recent research [8], [9], [18] proposed a self-supervised learning approach, to predict the next DTC in a sequence of DTC fault events, using neural embeddings and sequential models. These models solve the problem of fault event representation as well as eliminate the reliance on the repair data, which constrains the problem formulation to classification tasks only. In order to evaluate the effectiveness of these models, top-3 or top-5 accuracy of a single attribute, or multiple attributes per timestep is used as a metric. For example, the DTCEncoder [9] was able to achieve 79% top-5 accuracy.

In this paper, we combine Transformer [21] and GRU models [3] to boost the performance of the next DTC prediction task. We applied the transformer layer to learn the representation of DTC events before passing it to a single GRU layer and witnessed a 2% increase in the top-5 accuracy benchmark of the next-DTC prediction task. Our proposed model achieves a top-5 accuracy of 81.4% and a loss of 4.33, which is better than the standalone transformer and recurrent neural network models. We believe that the combined embeddings of all three attributes reflect in complex dependencies, which benefit from the transformer's ability to examine the context at a particular timestep against events at different timesteps,

SAC'24, April 8 –April 12, 2024, Avila, Spain
© 2024 Association for Computing Machinery.
ACM ISBN 979-8-4007-0243-3/24/04…$15.00
https://doi.org/xx.xxx/xxx_x

in multiple ways with the help of multiple heads. Moreover, GRU reinforces the model's understanding of the temporal dynamics in the sequence.

The rest of the paper is structured as follows: in section 2 we share the background and related work. Section 3 provides details about the methodology used by DTC-TranGru for the next DTC prediction task. The experiments and results are presented in section 4. We close the paper with a discussion of our approach and future work.

2 BACKGROUND AND RELATED WORK

In this section, we first present predictive approaches that use DTC event data and then provide a brief introduction to the self-supervised learning approach to predict the next DTC fault event, particularly in the absence of repair and warranty data. We conclude this section by highlighting research using this self-supervised next DTC prediction approach.

Before the shift towards the usage of OBDs, researchers mainly focused on using data retrieved from numerous sensors placed in the vehicle. The availability of the DTC fault event dataset sparked an interest in using it to perform different tasks relating to predictive maintenance. For example, Random Forest [2] has been employed to classify the DTC sequences into faulty and non-faulty classes [17]. [22] also used Random Forest along with AdaBoost [20] to predict the status of a single component, namely the starter motor, using DTCs recovered from specific modules only. [5] used Associative Classification [13] to learn patterns, which can help to flag if DTC events in the sequence are the actual fault events generated in the car or are the one that was raised as a result of experimentation by engineers in the workshop.

Instead of casting the problem into a classification task, or limiting the scope to a particular component, a self-supervised next-DTC prediction task has been proposed recently [8]. In such approach, we consider a sequence S of all DTC events associated with a vehicle and take the last N DTC events as training examples to predict the event at the next timestep, i.e., N+1. Each event e_t at timestep t has multiple attributes, each of which provides different information about the nature, granularity, and location of the fault. For instance, ECU provides information about which particular electric control unit has generated the fault, while base-DTC is the actual diagnostic trouble code, and fault-byte can be thought of as the most granular information about the fault, for example, chip-level information.

Figure 1 uses a simplified example of a DTC sequence to show the next-DTC prediction task, where the first-time step reports an event coming from the Telemetry ECU module (TM), and Battery as base-DTC, and 31 as fault byte. Subsequent events are recorded according to the same 3 attributes up to timestep t. The task is to predict the attributes for the next event. Since these attributes are categorical, with a high number of categories (for example 419 categories for base-DTC attribute), it is not efficient to apply conventional approaches, like One-Hot Encodings (OHE) to represent these events. Recent self-supervised models such as [8], [9], [18] use neural embedding layers to learn a dense representation that is substantially lower than the count of distinct classes in each attribute, hence facilitating the prediction of the next DTC. These

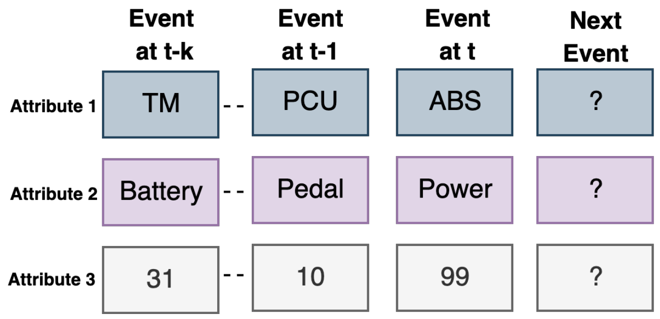

Figure 1: A DTC fault event sequence with N DTC events. Each event from time t-k to time t has three attributes. The goal is to predict the next DTC event at time $t + 1$ with all three attributes.

independent attribute level embedding layers are concatenated and learned along with the prediction task.

The self-supervised next-DTC prediction approach was first introduced as the SMFP model in [8], which uses neural embeddings [1] and a LSTM neural network to predict the next fault sequence. DTCEncoder [9] is another proposal benefitting from this approach: it uses Loung attention [14] before the GRU layer and then passes the GRU output through a dense bottleneck layer, which produces a compact representation that is used to perform approximate-neural-neighbour search and interpretation. The Transformer-decoder [18] approach uses a small-GPT-2 [19] architecture along with embeddings to perform the DTC prediction task.

3 METHODOLOGY

This section provides details of the proposed architecture, which combines the Transformer and the GRU layer to predict the next event with all three attributes per timestep.

3.1 DTC-TranGru Model

The overall architecture of the DTC-TranGru model is shown in figure 2 and the pseudocode is provided in algorithm 1. It is composed of the following main components.

3.1.1 Embedding Layer. Due to the high cardinality of DTC event's attributes, recent research [8], [9] has used neural embeddings to learn low-dimensional representations of attributes. In our model, we start with creating independent embedding layers for each input attribute to capture the semantic representation of the input tokens. These embedding layers are then concatenated along the feature dimension to create a unified embedding representation. In the DTC-TranGru architecture, before passing the combined embeddings to the next module, we apply a 1D spatial dropout, with a dropout rate of 0.1, to these combined embeddings.

3.1.2 Positional Encoding. The transformer layer, which will be detailed in 3.1.3, does not retain positional information of the sequence it works on. It hence needs some mechanism to pass the information relating to the temporal order of the events in the sequence. Researchers have introduced different positional encoding techniques [4], to be used with the transformer layers, to keep

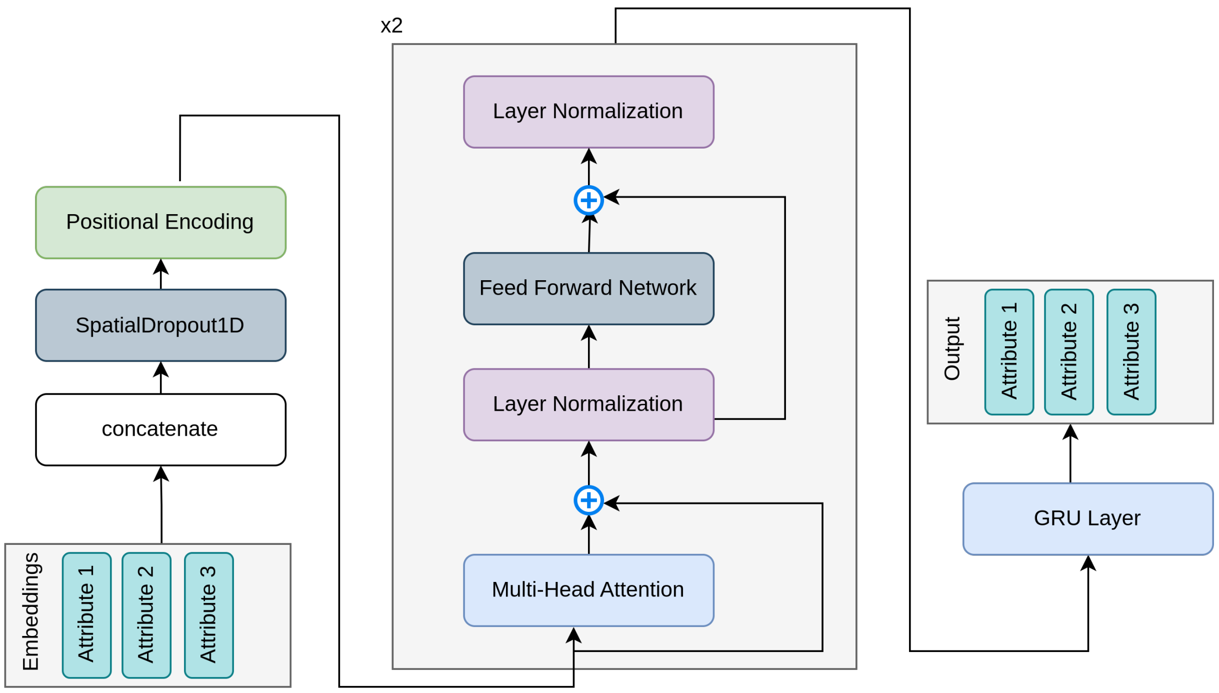

Figure 2: Architecture diagram for the DTC-TransGru. The left side of the figure shows pre-transformer operations, where DTC-TransGru starts by applying separate embedding layers to each attribute, and concatenates the individual embedding layers before passing them to spatial-1d dropout. In the next step, positional encoding is calculated on the embeddings before piping them to the two consecutive encoder layers of the transformer. A transformer encoder block, which has two encoder layers, is shown in the middle of the figure and the right side of the figure depicts post-transformer operations. The transformer encoder layer is followed by a GRU layer, before being passed to each individual dense softmax output layer.

particulars about the order dynamics. Hence, we pass the output of the spatial dropout layer to the positional encoding layer.

We use the same positional encoding approach that was used in the original transformer model [21], which utilizes sine (sin) and cosine (cos) functions to create position embeddings for each token in a sequence. For each position t, it computes $sin(t/10000^{(2d/T)})$ and $cos(t/10000^{(2d/T)})$ to generate distinct position embeddings, where d is the embedding dimension and T is the sequence length.

3.1.3 Transformer Layer. As opposed to SMFP [8], instead of passing embeddings directly to the recurrent layer, we set a transformer layer in front of the recurrent layer. In order to achieve that, we first pass the concatenated embeddings to the positional encoding layer followed by the transformer layer, which captures different complex contextual dependencies in the input sequence. The workings of the transformer layer are briefly described below.

Apart from the need for positional encoding, the standard transformer model employs multiple encoder layers, where each encoder layer is comprised of multi-head attention, residual-feed-forward dense layers, and layer-normalization. Depending on the use case, it may also use multiple decoder layers. The encoder layers process the input sequence to generate contextualized representations, while the decoder layer consumes these representations along with its previous output, to generate the next output in the sequence. The attention mechanism used in the encoder and decoder layers is different, where the encoder uses multi-head self-attention and the decoder uses masked self-attention.

The self-attention mechanism in the transformer model works by taking three inputs, i.e., the query matrix (Q), the key matrix (K), and the value matrix (V). The query matrix (Q) represents the current token for which the model wants to find the relevant information in the input sequence, whereas the key matrix (K) behaves as a memory by holding the tokens in the input sequence to look up relevant information. Finally, the value matrix (V) maintains the associated features (i.e., values) for each token in the input sequence.

If we consider d_k to be the number of dimensions used to represent each *key* in the key matrix K, we can write the attention mechanism in the transformer formally as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

By learning a weight matrix for each of these, we can perform a scaled dot-product attention mechanism, which corresponds to a single *head* in a multi-head attention mechanism. A single head, say i_{th} head in multi-head attention can be written as

$$\text{head}_i = \text{Attention}(QW_{Qi}, KW_{Ki}, VW_{Vi}) \quad (2)$$

Within the multi-head attention of the encoder layer, the input sequence is simultaneously analyzed by several attention heads. This allows the model to grasp various interdependencies among tokens and also allows computation to be performed in parallel, hence making it computationally more efficient. Now, we can represent multi-head attention with the help of the learned matrix W_O as follows

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_n)W_O \quad (3)$$

In neural networks having a large number of layers, gradients often become extremely small as they are propagated backward through layers, resulting in a problem called vanishing gradient. Residual connections make training deep models easier by ensuring that gradients can flow smoothly, preventing issues like vanishing gradients, and allowing deep networks to learn effectively. Similarly, a layer-normalization technique makes sure that the values passed between layers aren't too extreme, enabling smoother gradients, faster training, and better generalization accuracy. As shown in figure 3, which depicts the detailed view of an encoder layer of the DTC-TranGru's transformer layer, the output from multi-head attention undergoes layer-normalization operation along with residual connection.

Subsequently, the output proceeds through a Feed-Forward Neural Network (FFN), which employs two sequential linear transformations separated by a ReLU activation, before going through another residual connection and layer-normalization operation. The FFN in the transformer model can be represented as:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2 \quad (4)$$

The first layer in FFN takes the first layer-normalization's output and projects it up to 256 dimensions. In order to go through the second residual connection, which requires the addition of the first layer normalization layer and the second FF1 dense layer, the second layer reduces the dimension down to 38 (concatenated embedding size).

This comprehensive process executed within the encoder layer effectively captures local and global relationships between tokens, facilitating the learning of meaningful representations.

In DTC-TranGru, we used two such encoder layers and 4 attention heads to learn the representation of the DTC sequence.

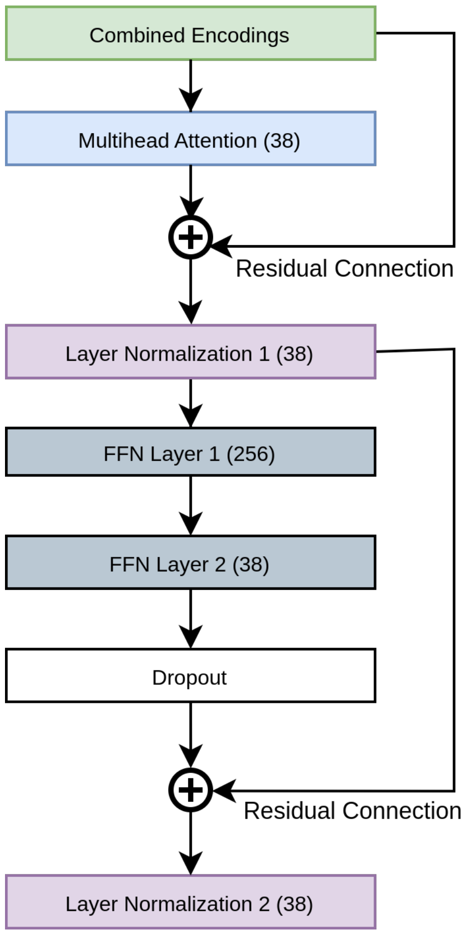

Figure 3: Detailed view of the encoder layer of the DTC-TranGRU's transformer block. Each layer shows its output dimension next to its name. It can be seen that there are two residual connections, two dense layers in FFN, and two layer-normalization layers. The dimension of the output is scaled up to 256 in the first dense layer, FF1, and, to perform the second residual addition, it is scaled down to the size of the combined embeddings (38) in FF2.

3.1.4 GRU Layer. As shown in figure 4, the transformer layer produces an output that has N timesteps, where each timestep has a dimensionality equal to *EMB-SIZE*. To make the output of the transformer compatible with the dense output layer, researchers typically remove the time dimension by either summing, averaging, or applying 1-D Global average pooling to the output of the transformers.

As shown in figure 4, unlike the conventional approaches, which work by getting rid of the time dimension, we pass the output of the transformer layers to a Gated Recurrent Unit (GRU) network.

A GRU layer is a type of recurrent neural network (RNN) that is designed to counteract the vanishing gradient issue. It incorporates two gates, the reset gate, and the update gate, which play a pivotal role in regulating the internal information flow of the unit. The reset gate in the GRU helps to decide what information from the recent time steps to forget, whereas the update gate controls the threshold of the new information to add. Through this mechanism of gate control, GRUs effectively capture distant dependencies in sequences, all while maintaining computational efficiency.

The way GRU works, it incorporates all temporal dynamics of the sequence and summarizes the information of all preceding timesteps into the current hidden state. In the DTC-TranGRU, this last state of the GRU layer is extracted and passed to separate output softmax layers for each attribute, enabling the prediction of the next event. Applying the GRU layer not only allows us to pass the output of the transform to the output layers but also learns contextual dependencies associated with the events.

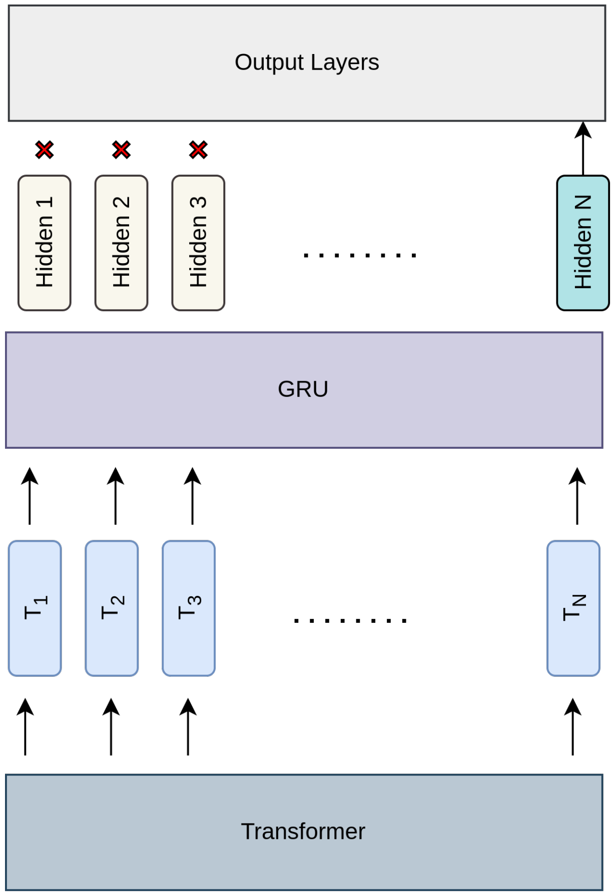

Figure 4: The transformer layer returns N timestep outputs (each containing *EMB-SIZE* latent dimensions), which typically are averaged, summed, or go through the GlobalAveragePooling1D layer to make it compatible with the dense output layer. Instead of doing this, we applied a GRU layer on top of the transformer, where each N_i dimension in the N dimensional transformer is passed to the i^{th} timestep of the GRU. Since the last hidden state of the GRU incorporates all the latent information about the previous timesteps, it is passed to the individual dense output layers of each attribute.

Algorithm 1 DTC-TranGru

```
Require: $seq_1 \dots seq_N$
Ensure: DTC event ($attr^1$,$attr^2$,$attr^3$) for $seq_1 \dots seq_N$
  for $epoch \leftarrow 1$ to $N$ do
    $a^1_{EMB} \leftarrow EMB(a^1_{OHE})$
    $a^2_{EMB} \leftarrow EMB(a^2_{OHE})$
    $a^3_{EMB} \leftarrow EMB(a^3_{OHE})$
    $a_{EMB} \leftarrow CONCAT(attr^1_{EMB}, attr^2_{EMB}, attr^3_{EMB})$
    $gru\_state \leftarrow 1DSpatialDropout(a_{EMB})$
    $pos\_enc \leftarrow POSITIONAL\_ENCODING(a_{EMB})$
    $enc \leftarrow pos\_enc$
    for $encoder\_layer \leftarrow 1$ to $N$ do
      $multihead \leftarrow multihead_n(enc)$
      $l\_norm\_1 \leftarrow Layer\_Norm(pos\_enc + multihead)$
      $ffn\_l1 \leftarrow Dense(multihead)$
      $ffn\_l2 \leftarrow Dense(ffn\_l1)$
      $ffn\_l2 \leftarrow Dense(ffn\_l2)$
      $l\_norm\_2 \leftarrow Layer\_Norm(ffn_l2 + l\_norm\_1)$
      $enc \leftarrow l\_norm\_2$
    end for
    $GRU\_OUTPUT \leftarrow GRU(enc)$
    $attr^1_{pred} \leftarrow Dense(GRU\_OUTPUT)$
    $attr^2_{pred} \leftarrow Dense(GRU\_OUTPUT)$
    $attr^3_{pred} \leftarrow Dense(GRU\_OUTPUT)$
    Calculate loss
    Optimize parameters of all layers
  end for
```

4 EXPERIMENTS AND RESULTS

This section starts with providing details about the DTC event dataset used for training DTC-TranGru and of the experimental setup. It concludes by reflecting on the results of all experiments performed.

4.1 Dataset and Data Preprocessing

This research used vehicular DTC sequence data provided by *Name removed for the blink-review.* The dataset comprises of a total 250,000 sequences, where each sequence belongs to a unique vehicle and has three attributes at each timestep.

For each DTC sequence, the events were first ordered by the time of occurrence. Individual attributes for each sequence are vectorized and tokenized separately. This preprocessing step is shown in figure 5, which manifests 2 DTC sequences before and after the pre-processing step. There are 83 unique classes in the first attribute (ECU), 419 in the second attribute (base-dtc), and 64 in the third attribute (fault-byte).

All sequences were restricted to the last N DTC events, and those having less than N DTC events were padded with a special token (‘0‘). We then split the data into validation, test, and training sets. Out of 250,000 sequences, we kept 12500 sequences separate for testing, 4750 for validation, and used 232,750 event sequences for training the DTC-TranGru.

4.2 Experimental setup and hyperparameter tuning

The hyperparameters and parameter choices, which are shown in table 1, are selected by running the Hyperband [12] hyperparameter tuning method available in the python keras-tuner [16] library.

The major parameter to consider for the transformer layer was the number of heads used in the multi-head attention layer, for which we obtained the best performance with 4 heads, while in the GRU layer, 128 GRU units showed the best result. Placing dropouts, including recurrent-dropout, did not make much of a difference in the GRU layer in our experiments. However, introducing a 1D spatial dropout after embedding layers provided slightly better results. A learning rate of 0.0052 provided the best result using the Adam optimizer [11].

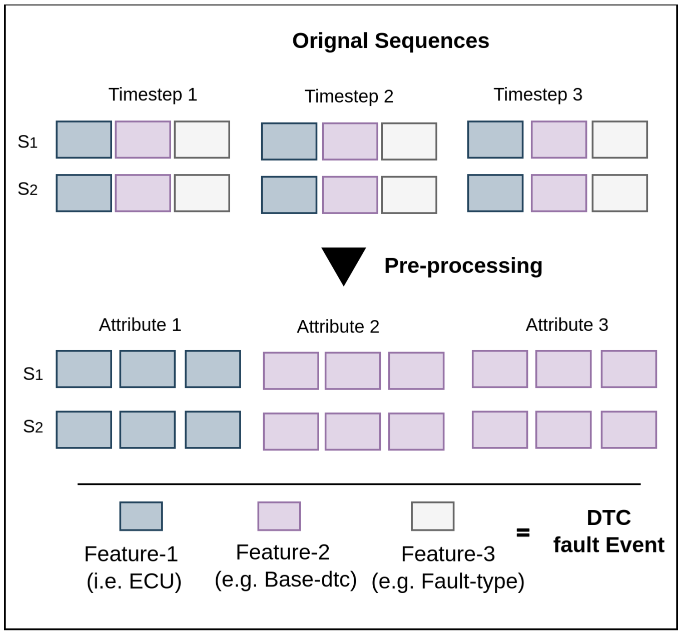

Figure 5: An example of two DTC fault sequences with 3 DTCs each, undergoing the preprocessing step. Each attribute is vectorized and separated so that it can then be passed to its independent embedding layer. The choice of the number of events is just for the sake of illustration, otherwise, all of the DTC sequences used in this experiment consist of 5 DTC events at least.

For the first dense layer of the FFN in the encoder layer, we tried dimension sizes between 96 and 256 with hyperparameter tuning and found that 256 neurons have the best results. Since the second residual connection in the encoder layer adds the output of the second FFN dense layer to the output of the first normalization layer, it constraints the dimension of the second dense layer, which requires it to be equal to the size of the combined encoding. Hence, there is no need to experiment with the size of the second dense layer.

We used RELU [15] as an activation function in the GRU layer. For the dense output layers, the softmax activation function was used to provide a probability of occurrence of each DTC event's attribute.

To calculate the loss and performance of the model, we used cross-categorical loss for the output layer of each attribute and summed the individual loss of all three attributes as follows

$$L(\hat{y}, y) = \sum_{a^i}^{A}(-\sum_{k}^{K_{a^i}} y^{(k)} \log(\hat{y^{(k)}})), \tag{5}$$

where K_{a^i} is the number of unique classes in a given attribute a_i, A represents the number of total attributes, $\hat{y}$ represent the predicted class, and y denotes the actual class.

Table 1: Parameters and hyper-parameter choices along with the selected values. The main parameters to consider for DTC-TranGru were the total number of heads in the multi-head attention mechanism and the number of encoder layers to use in the transformer. The main hyperparameter choice corresponded to the selection of the appropriate learning rate.

Choice	min	max	final
Learning rate	1e-4	0.1	0.005
Number of heads	1	6	4
Number of encoder layers	1	5	2
FFN Dimension	128	256	256
Attribute-1 (module) embedding	4	24	6
Attribute-2 (base-dtc) embedding	12	56	24
Attribute-2 (fault-byte) embedding	4	32	8
Spatial dropout after embeddings	0.0	0.5	0.1
GRU layer units	96	256	128

4.3 Results

Table 2 shows the results of the ablation study and the comparison of the DTC-TranGru model with other next-DTC prediction models. We can see that the DTC-TranGru achieved better results than all other models including the DTCEncoder [9], which uses the attention mechanism along with the GRU layer to predict the next DTC in the sequence.

DTC-TranGru was also compared with the standalone transformer model, and the LSTM-based SMFP model [8] as a part of the ablation study. Table 2 shows that the DTC-TranGru performs better than these standalone models. We argue that the standalone transformer model works well with generative approaches, where there is an abundance of training data and variety in the output is considered beneficial. In the problem at hand, however, we do not have a large amount of data available and it is required to strictly follow the sequential order for the prediction of the next DTC event. The recurrent nature of GRU's hidden states incorporates precise contextual dependencies.

To test if having a GRU layer alone after the transformer can keep positional information intact, we experimented with removing the positional encoding layer from the transformer. However, the accuracy of the model decreased with the removal of the positional encoding layer. It indicates that although a GRU layer reinforces sequential information, it is still relevant to have positional encoding in the transformer layer.

We also compared the result of DTC-TranGru with and without the 1D-spatial dropout layer, which was applied after the concatenated embeddings layer. As shown in table 2, we witness a slight increase in the performance of the model with the usage of 1D-spatial dropout. We assume that employing this dropout layer after combined embeddings forces the model to learn generalizable representations for event attributes and reduces over-reliance on specific combinations of attributes.

Given the size of the dataset and the nature of the task, it was obvious that the Transformer-Decoder model [18], which is based on a smaller version of the GPT-2 model [19] (often referred to as

Table 2: Comparison of the results achieved by DTC-TranGru compared with SMFP [8], DTCEncoder [9], [18] and standalone models. Results for DTC-TranGRU without the 1D-spatial dropout layer after concatenated embeddings and without positional encodings are also compared. The best results, achieved by DTC-TranGru, are highlighted in bold.

Architecture	Validation Loss	Top-5 Test Accuracy
SMFP	4.50	76.15%
DTCEncoder	4.36	79.21%
Transformer Only	4.47	78.3%
Transformer-Decoder (small-GPT2)	7.6	40%
DTC-TranGru (w/o positional-encoding)	4.49	78.5%
DTC-TranGru (w/o 1D-spatial dropout)	4.35	79.5%
DTC-TranGru	**4.33**	**81.4%**

small-GPT-2), would overfit. But for the sake of completeness and accurate comparison, we modified the model used in the Transformer-Decoder [18] by reducing the number of encoder layers from 12 to 6 and of FFN dense layer neurons from 1024 to 512. This modification was done due to computation constraints and to reduce the overall parameters in the original model. As seen in table 2, the resulting model overfits very early and hence underperforms the DTC-TranGru model by quite a large margin. It achieves a top-5 accuracy of 40% only, with a validation loss of 7.6. As discussed previously, this might be due to model overfitting and being overly complex for the task and size of the dataset.

5 DISCUSSION AND CONCLUSION

In this paper, we present a transformer and GRU-based architecture to improve the performance of the next DTC prediction task. In the proposed architecture, which we call DTC-TranGru, instead of using standalone recurrent or transformer models, we combine these individual models by passing the encoding learned by the transformer to the GRU layer. We show that our model provides better results as compared to the standalone models and improves the top-5 prediction accuracy benchmark by 2%, as it achieves a top-5 accuracy of 81.4% and reduces validation loss to 4.33, where the prediction of the next DTC includes predicting three different attributes for the next DTC prediction.

We believe that the self-attention mechanism and the encoder layer in the transformer help DTC-TranGru learn to represent the DTC sequence in a way that incorporates multiple hidden patterns among the DTC faults, with the help of multiple heads. Furthermore, the GRU layer takes the representation learned by the transformer to strengthen the contextual and order semantics of the DTC sequence. Combined, both these models provide DTC-TranGru with the ability to understand the context better for each DTC via encoding learned by the transformer and simultaneously taking into account the sequential dependencies in the form of hidden states of GRU.

The DTC-TranGru and related recent approaches have opened the door to decode the complex dependencies and patterns in the sequence of events, both in terms of representation and modeling. The DTC-Encoder specifically shows that with smaller and domain-specific datasets, a small transformer network with few encoder layers can learn robust representations in contrast to a huge model. Furthermore, these presentations learned by the transformer's encoder layer might further be used by different networks suitable to the problem. We believe that with the availability of more data, and the ability to incorporate metadata about vehicles and their conditions, there will be a chance for researchers to better predict the next possible faults expected in the vehicles and predict the need for maintenance ahead of time.

ACKNOWLEDGMENTS

This research is supported by *acknowledgements removed for the blink-review.*

REFERENCES

[1] Y. Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A Neural Probabilistic Language Model. *Journal of Machine Learning Research* 3, 932–938. https://doi.org/10.1162/153244303322533223

[2] L Breiman. 2001. Random Forests. *Machine Learning* 45 (10 2001), 5–32. https://doi.org/10.1023/A:1010950718922

[3] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:1406.1078 [cs.CL]

[4] Philipp Dufter, Martin Schmitt, and Hinrich Schütze. 2022. Position Information in Transformers: An Overview. *Computational Linguistics* 48 (06 2022), 1–31. https://doi.org/10.1162/coli_a_00445

[5] Moa Fransson and Lisa Fåhraeus. [n. d.]. Finding Patterns in Vehicle Diagnostic Trouble Codes : A data mining study applying associative classification. http://uu.diva-portal.org/smash/get/diva2:828052/FULLTEXT01.pdf

[6] Flavio Giobergia, Elena Baralis, Maria Camuglia, Tania Cerquitelli, Marco Mellia, Alessandra Neri, Davide Tricarico, and Alessia Tuninetti. 2018. Mining Sensor Data for Predictive Maintenance in the Automotive Industry. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, Turin, Italy, 351–360. https://doi.org/10.1109/DSAA.2018.00046

[7] Hardik A. Gohel, Himanshu Upadhyay, Leonel Lagos, Kevin Cooper, and Andrew Sanzetenea. 2020. Predictive maintenance architecture development for nuclear infrastructure using machine learning. *Nuclear Engineering and Technology* 52, 7 (2020), 1436–1442. https://doi.org/10.1016/j.net.2019.12.029

[8] AB Hafeez, E Alonso, and A Ter-Sarkisov. 2021. Towards Sequential Multivariate Fault Prediction for Vehicular Predictive Maintenance. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, Pasadena, CA, USA.

[9] Abdul Basit Hafeez, Eduardo Alonso, and Atif Riaz. 2022. DTCEncoder: A Swiss Army Knife Architecture for DTC Exploration, Prediction, Search and Model Interpretation. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, Nassau, Bahamas, 519–524. https://doi.org/10.1109/ICMLA55696.2022.00085

[10] Shashidhar Kaparthi and Daniel Bumblauskas. 2020. Designing predictive maintenance systems using decision tree-based machine learning techniques. *International Journal of Quality Reliability Management* ahead-of-print (02 2020). https://doi.org/10.1108/IJQRM-04-2019-0131

[11] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (2014).

[12] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* 18, 185 (2018), 1–52. http://jmlr.org/papers/v18/16-558.html

[13] Bing Liu, Wynne Hsu, and Yiming Ma. 1970. Integrating Classification and Association Rule Mining. *Proceedings of 4th International Conference on Knowledge Discovery Data Mining (KDD)* (02 1970).

[14] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. https://doi.org/10.48550/ARXIV.1508.04025

[15] Vinod Nair and Geoffrey Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair. *Proceedings of ICML* 27, 807–814.

[16] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. 2019. KerasTuner. https://github.com/keras-team/keras-tuner

[17] Parivash Pirasteh, Slawomir Nowaczyk, Sepideh Pashami, Magnus Löwenadler, Klas Thunberg, Henrik Ydreskog, and Peter Berck. 2019. 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* (Melbourne, VIC, Australia) *(WIDM'19)*. Association for Computing Machinery, New York, NY, 10 pages.

[18] H. Poljo. 2021. *Transformer decoder as a method to predict diagnostic trouble codes in heavy commercial vehicles.* Master's thesis.

[19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[20] Robert E. Schapire. 2013. Explaining AdaBoost. In *Empirical Inference.* https://api.semanticscholar.org/CorpusID:7122892

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[22] Linda Virkkala and Johanna Haglund. [n. d.]. *Modelling of patterns between operational data, diagnostic trouble codes and workshop history using big data and machine learning.* Ph. D. Dissertation. Uppsala universitet. https://www.diva-portal.org/smash/get/diva2:909003/FULLTEXT01.pdf

[23] Zhongju Zhang and Pengzhu Zhang. 2015. Seeing around the corner: an analytic approach for predictive maintenance using sensor data. *Journal of Management Analytics* 2, 4 (2015), 333–350. https://doi.org/10.1080/23270012.2015.1086704

[24] Pushe Zhao, Masaru Kurihara, Junichi Tanaka, Tojiro Noda, Shigeyoshi Chikuma, and Tadashi Suzuki. 2017. Advanced correlation-based anomaly detection method for predictive maintenance. In *2017 IEEE International Conference on Prognostics and Health Management (ICPHM).* 78–83. https://doi.org/10.1109/ICPHM.2017.7998309