UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

Harnessing event sensory data for error pattern prediction in vehicles: a language model approach

Hugo Math, Rainer Lienhart, Robin Schön

Angaben zur Veröffentlichung / Publication details:

Math, Hugo, Rainer Lienhart, and Robin Schön. 2025. "Harnessing event sensory data for error pattern prediction in vehicles: a language model approach." *Proceedings of the AAAI Conference on Artificial Intelligence* 39 (18): 19423–31. https://doi.org/10.1609/aaai.v39i18.34138.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

Harnessing Event Sensory Data for Error Pattern Prediction in Vehicles: A Language Model Approach

Hugo Math, Rainer Lienhart, Robin Schön

Augsburg University, Augsburg 86159, Germany

Abstract

In this paper, we draw an analogy between processing natural languages and processing multivariate event streams from vehicles in order to predict *when* and *what* error pattern is most likely to occur in the future for a given car. Our approach leverages the temporal dynamics and contextual relationships of our event data from a fleet of cars. Event data is composed of discrete values of error codes as well as continuous values such as time and mileage. Modelled by two causal Transformers, we can anticipate vehicle failures and malfunctions before they happen. Thus, we introduce *CarFormer*, a Transformer model trained via a new self-supervised learning strategy, and *EPredictor*, an autoregressive Transformer decoder model capable of predicting *when* and *what* error pattern will most likely occur after some error code apparition. Despite the challenges of high cardinality of event types, their unbalanced frequency of appearance and limited labelled data, our experimental results demonstrate the excellent predictive ability of our novel model. Specifically, with sequences of 160 error codes on average, our model is able with only half of the error codes to achieve 80% F1 score for predicting *what* error pattern will occur and achieves an average absolute error of 58.4 ± 13.2h *when* forecasting the time of occurrence, thus enabling confident predictive maintenance and enhancing vehicle safety.

Code — https://github.com/Mathugo/AAAI2025-CarFormer-EPredictor

1 Introduction

Today's vehicles generate an astounding amount of data, typically reported as events on an irregular basis, but continuously over time. Some events occur simultaneously, while others are scattered over time, with unequally distributed time intervals between their occurrence. They usually report numerical and/or categorical features. In our work, we focus on processing and analyzing such *multivariate and irregular event streams* produced by modern cars, for which related research is still sparse. Our sequences of discrete events in time are known as DTCs (*diagnostic trouble codes*). DTCs are preferred over raw sensory data because they provide less granular and discrete information. This makes them easier to analyze.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

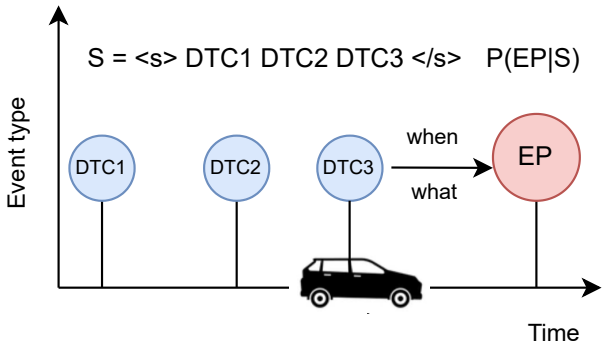

Figure 1: Error pattern (EP) prediction (when and what with which probability) based on the past sequence S of diagnostic trouble codes (DTCs).

Our goal is to learn the correlation between DTCs using a DTC-based language model to predict *when* and *what* with which *probability* an error pattern (EP) is occurring after having seen a number of DTCs (see Figure 1). Thus, we consider DTCs to be the words in our language. In contrast, EPs are very different from DTCs. They are defined by domain experts after observing DTC sequences. Therefore, they are way more precise about the critical error that the car is having. While some DTCs can also be noisy and repetitive events about recurring errors (e.g. electrical issues, software updates), EPs characterize a whole error sequence consisting of precise vehicle failures (e.g. engine or battery failures). Recent research has approached predictive maintenance in the automotive field using sequences of DTCs via RNNs (Hafeez, Alonso, and Ter-Sarkisov 2021) and, more recently, Transformers (Hafeez, Alonso, and Riaz 2024) to predict the next DTC. However, distinguishing minor errors or noise-like DTCs from important events such as error patterns (EPs) is essential since the latter poses higher risks and necessitates greater safety and maintenance measures, such as vehicle immobilization or addressing critical malfunctions. Furthermore, as the data volume increases, accurately predicting the next DTC becomes challenging, particularly when the event type cardinality approaches $\sim 10^4$. This phenomenon is akin to language processing, where the accuracy of the next token prediction using greedy decoding or other

methods exponentially decreases with sequence length and vocabulary size due to error accumulation (Bachmann and Nagarajan 2024). Historically, Hawkes Processes and their neural variants have advanced the state of the art in event modelling for next event and time prediction tasks (Hawkes 1971; Du et al. 2016; Shchur et al. 2021). Transformer-based models like BERT (Devlin et al. 2019) and GPT-3 (Brown et al. 2020) have gained overwhelming popularity due to their attention-based architecture, flexibility, parallelization, and state-of-the-art performance in sequence modelling. Consequently, models adapted to discrete-time sequences using Transformers have emerged naturally (Zhang et al. 2020a; Zuo et al. 2020; Shou et al. 2024), achieving state-of-the-art performance in next event prediction benchmarks.

By leveraging the sequential nature of our data, we can define sentences as the concatenation of each discrete event:

$$" < s > \ DTC1 \ DTC2 \ .. \ DTCn \ < /s > "$$

as shown in Figure 1, and embed it into $\mathbb{R}^D$. We make several modifications to the vanilla Transformer from (Vaswani et al. 2017), incorporating continuous-time and mileage positional embeddings as additional context. Using two distinct training phases, we introduce CarFormer a pre-trained model acting as an encoder and EPredictor a decoder Transformer-based model that generates a probability distribution over a set of error patterns for each event step i to determine *what* EPs will most likely happen and estimates a time for *when* it will occur.

Contributions

To the best of our knowledge, introducing a causal Transformer model for error pattern and DTC prediction (event type and time) has not been explored, despite some related work on event-data and next-DTC prediction (Shou et al. 2024; Hafeez, Alonso, and Riaz 2024; Hafeez, Alonso, and Ter-Sarkisov 2021). The main contributions of this paper are:

- **CarFormer**: An encoder Transformer-based model designed to ingest scattered continuous event streams from vehicles, trained via a multi-task learning strategy. This model will transform the DTC-sequences into hidden representations that can be processed by the EPredictor.
- **EPredictor**: An autoregressive decoder Transformer-based model that specializes in predictive maintenance, particularly error patterns by estimating *when* and *what* error patterns will most likely occur.

2 Background and Related Work

Event Sequence Modelling with TPP

Event data is commonly modelled via Temporal Point Processes (TPPs), which describe stochastic processes of discrete events. Each event is composed of a time of occurrence $t \in \mathbb{R}^+$ and an event type $u \in U$ forming a pair (t, u). U is a finite set of discrete event types. A sequence is constructed with multiple pairs of events, such as $S = \{(t_1, u_1), ..., (t_L, u_L)\}$ where $0 < t_1 < ... < t_n$.

TPPs are usually represented as a counting process $N(t)\ \forall t \geq 0$ for the events of type u, which describes the number of occurrences of an event over time. The goal is to predict the next event (u', t') given the history $H_t := \{(t_i, u_i) \in \mathbb{R}^+ \times U | t_i < t\}$ of all events that occurred up to time t. We define λ^* to model the instantaneous rate of an event in continuous time. Thus, the probability of occurrence for an event (u', t') is conditioned on the history of events H_t:

$$\begin{aligned} \lambda^*(t)dt &:= P((u', t') : t' \in [t, t + dt) | H_t) \\ &= \mathbb{E}(dN(t)|H_t) \end{aligned} \tag{1}$$

which we could translate as the expected number of events during an infinitesimal time window $[t, t + dt)$ knowing the history H_t. We assume that two events do not occur simultaneously, i.e., $dN(t) \in \{0, 1\}$.

The Hawkes Process (Hawkes 1971) has been arguably the most studied modelling technique for TPPs. It assumes some parametric form of the conditional intensity function $\lambda^*(t)$ and states that an event excites future events additively and decays using a function f over time. For example:

$$\lambda^*(t) = \mu + \sum_{(u_i, t_i) \in H_t} f(t - t_i) \tag{2}$$

where $\mu \geq 0$ is the base intensity. Neural TPPs, on the other hand, aim at reducing the inductive bias of the Hawkes Process, which states that past events excite future ones additively. They do this by approximating λ^* using a neural network (RNN, LSTM, Transformer) (Zhang et al. 2020a; Du et al. 2016). Recent papers suggest using a generative and contrastive approach for event sequence modeling, showing promising results across predictive benchmarks. (Lin et al. 2022) uses next-event prediction as their main training objective, while (Shou et al. 2024) have three specific objectives plus a contrastive loss.

Transformers for Event Streams

Encoding the history H_t into historical hidden vectors using a Transformer enhanced the performance on event prediction benchmarks as shown in (Zuo et al. 2020) with the Transformer Hawkes Process (THP) or the self-attentive Hawkes Process (Zhang et al. 2020a). They typically reused the vanilla Transformer (Vaswani et al. 2017) and created two embeddings:

Time Embedding Time embedding replaces the traditional positional encoding which grants the Transformer model positional information of each token within the sequence. This time embedding is defined deterministically with periodic functions exactly like in (Vaswani et al. 2017):

$$\mathbf{P}_{i,j} := \begin{cases} \sin(t_i \times \omega_0^{j/d}) & \text{if } j \mod 2 = 0 \\ \cos(t_i \times \omega_0^{(j-1)/d}) & \text{if } j \mod 2 = 1 \end{cases} \tag{3}$$

where i is the index of the i-th event, ω_0 is the frequency (usually 10^{-4}).

Event-Type Embedding To get a dense representation of our sequence, we embed each event into a d dimensional space using an embedding matrix $\mathbf{L}^{V \times d}$ where V is the distinct number of events (vocabulary). As we would do for

word embeddings, we create a sequence of one-hot encoded vectors from the event types $\{u_i\}_{i=0}^{L}$ as $\mathbf{Y} \in \mathbb{R}^{L \times V}$. Thus, the event-type embedding $\mathbf{E} = \mathbf{YL} \in \mathbb{R}^{L \times d}$ and the input embedding $\mathbf{U}$ is defined as $\mathbf{U} = \mathbf{E} + \mathbf{P} \in \mathbb{R}^{L \times d}$

Attention The majority of research utilizing Transformer models for sequence data employs the architecture introduced by (Vaswani et al. 2017). We define three linear projection matrices $\mathbf{Q} = \mathbf{UW}^Q$, $\mathbf{K} = \mathbf{UW}^K$, and $\mathbf{V} = \mathbf{UW}^V$. They are called *query*, *key*, and *value*, respectively. $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$ are trainable weights. Essentially $\mathbf{Q}$ represents what the model is looking based on the input $\mathbf{U}$, $\mathbf{K}$ is the label for the input's information and $\mathbf{V}$ is the desired representation of the input's semantics. The attention score can be computed as:

$$\mathbf{A} = \text{softmax}(\mathbf{QK}^T/\sqrt{d}) \tag{4}$$

$$\mathbf{C} = \mathbf{AV} \tag{5}$$

where d denotes the number of attention heads and $\mathbf{A} \in \mathbb{R}^{L \times L}$ the attention scores of each event pair i, j. A final hidden representation H is obtained via a layer normalization (LayerNorm), a pointwise feed-forward neural network (FFN) and residual connections via:

$$\begin{aligned} \mathbf{U}' &= \text{LayerNorm}(\mathbf{C} + \mathbf{U}) \\ H &= \text{LayerNorm}(\mathbf{U}' + \text{FFN}(\mathbf{U})) \end{aligned} \tag{6}$$

Self-Supervised Learning

By leveraging an efficient pre-training task (e.g. token masking or next token prediction) and then fine-tuning a smaller model with fewer parameters for specific tasks, Transformer-based models achieve state-of-the-art performance in natural language processing tasks (Devlin et al. 2019; Brown et al. 2020). For example, the GPT model is pre-trained on the next token prediction task, where the labels are generated by shifting the tokens to the right. Then, a classification head is added on top of the Transformer model to assign a probability to each token. Contrary to BERT (Devlin et al. 2019) a causal mask is applied so that tokens can only attend to the previous one, thus preventing cheating. In section 4 we will introduce a pre-trained model serving as an encoder trained on specific event prediction tasks to ensure adaptability to the event stream domain of vehicles. We found that modelling event streams with autoregressive Transformer-based models for fault predictions were not addressed well in the literature (Shchur et al. 2021) although there is some related work for TPPs (Lin et al. 2022; Shou et al. 2024).

3 Data

Overview Diagnostic data is generated by various *Electronic Control Units* (ECU) in a vehicle at irregular intervals. Diagnostic data differs substantially from raw sensor data, since diagnostic data or fault events are categorical and relate to various problems within the vehicle. We construct a *Diagnosis Trouble Code* (DTC) indicating the precise error from 3 pieces of information arriving at the same timestamp ts and mileage d: (1) the ID number of the ECU, (2) an error code (*Base-DTC*) and (3) a *Fault-Byte*. A single DTC token is composed of these 3 elements:

$$DTC = ECU|\textit{Base-DTC}|\textit{Fault-Byte}$$

Thus, we can uniquely encode each DTC by a single token to predict the next token directly.

Notation	Description
u	Discrete event type, in our case a DTC.
d	Absolute mileage of the vehicle in km.
m	Mileage of the vehicle in km since the first DTC (u_0) occurred in a sequence S such as $m_i = d_i - d_0$
ts	Unix timestamp attached to each DTC.
t	Number of hours passed since the first DTC occurred in a sequence. More specifically, t_i is defined as $t_i = ts_i - ts_0$.
S	Sequence of triplets (event type, time, mileage) defined as $S = \{(u_i, t_i, m_i)\}_{i=0}^{L}$ of length L with index starting from 0.
$i \in \{0, ..., L\}$	index of element in (event) sequence.

Table 1: List of symbols and their respective meanings

This research uses an anonymized vehicular DTC sequence dataset of 1.7×10^6 sequences with on average 150 DTCs per sequence. Each sequence belongs to a unique vehicle. In a sequence S, each DTC (commonly referred to as event type u_i) is attached with a time t_i and a mileage m_i constructing a single event (u_i, t_i, m_i). You can find an overview of the DTC elements in Table 2.

Data	# of values	Description
DTC	8710	Diagnostic Trouble Code
ECU	61	Electronic Control Unit
Base-DTC	7726	Error Code
Fault-Byte	2	Binary Value

Table 2: Number of Distinct values of the DTC elements

To get a full sequence $S = \{(u_i, t_i, m_i)\}_{i=0}^{L}$, we obtain the last known timestamp ts_L and mileage d_L and select all DTCs that are no further than: (1) a given period in the past ($ts_L - ts_i \leq 30$ days) and (2) a given distance in the past ($d_L - d_i \leq 300$km). Table 1 explains the different data notation.

Time and Mileage We draw the distribution of t_i and m_i in Figure 2. We observe peaks at 0 on both distributions due to truncation and missing values. In order to feed our model with the time t feature, we need a scaling method to level out the left-tail distribution somewhat. Using $\log_b(t+1)$ is a natural choice. At the same time, we want to approximately map t into the range of $[-1, 1]$. Therefore, we apply the following non-linear function $f_t : \mathbb{R}^+ \to \mathbb{R}$ to t:

$$t' = f_t(t, b) = \log_b(t + 1) - 1 \; \forall t \in \mathbb{R}^+ \tag{7}$$

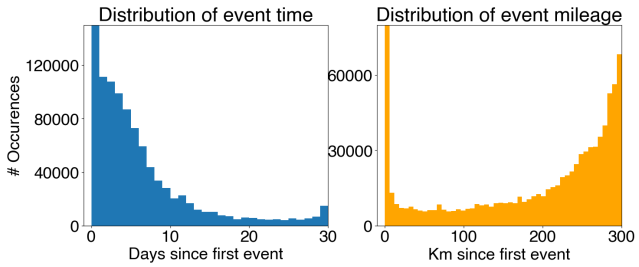

Figure 2: Distribution of t_i and m_i in our data set

with b chosen appropriately and feed t' into a neural network. In TPPs the time is usually represented as inter-event time or its logarithm (Shchur et al. 2021; Du et al. 2016).

4 Pre-training

Embeddings

CarFormer uses 4 different embeddings to capture spatial and temporal dependencies of irregular event apparition. These embeddings differ from the positional embedding $\mathbf{P}$ used in TPPs (2020), (2020b) and next-DTC prediction studies (2024), (2021). We embed both time t and mileage m and use a rotation matrix to induce absolute and relative event positions for our Transformer such as:

- **Event-type embedding** $\mathbf{E} \in \mathbb{R}^{L\times d}$ is obtained like described in section 2.
- **Absolute time embedding** $\mathbf{T} \in \mathbb{R}^{L\times d}$ is constructed on-the-fly at each forward pass by a linear transformation $t_{i,j} = t'_i w_j + b_j$ where w_j, b_j are learnable parameters and t'_i is the scaled time at event step i.
- **Mileage embedding** $\mathbf{M} \in \mathbb{R}^{L\times d}$ is obtained via a learnable lookup table $\mathbf{W}^{m_{\max}\times d}$ where $m_{\max} = 300$km. Each row $\mathbf{w}_m \in \mathbb{R}^d$ corresponds to the learnable embedding vector for the discrete mileage m. The continuous mileage $m_i \in \mathbb{R}^+$ is cast to an integer value $m = \lfloor m_i \rfloor$.
- **Rotary Position Embedding (RoPE)** $\mathbf{R}^d_\Theta$ Due to the permutation invariance of the Transformer model and the scattered time t, we still need to integrate positional event information. To do so $\mathbf{Q}, \mathbf{K}$ are rotated using the orthogonal matrix $\mathbf{R}^d_\Theta$ from (Su et al. 2024) in function of the absolute event position i in the sequence S. This method has two advantages: (1) it's not learnable (less likely to over-fitting), and (2) it integrates natively the relative position instead of altering $\mathbf{A}$ with a learnable bias like in (Shaw, Uszkoreit, and Vaswani 2018).

We make the distinction between our event-type embedding $\mathbf{E}$ and the other information per event (time and mileage) which we call context embedding $\mathbf{CE} = \mathbf{T} + \mathbf{M}$.

Continuous Time Mileage Aware Attention We modify the vanilla Transformer from (Vaswani et al. 2017) by (1) adding the context embedding to the projected event-type embedding at every layer (Touvron et al. 2023), and then (2) a Rotary Position Embedding (RoPE) (Su et al. 2024) is applied to both query $\mathbf{Q}$ and key $\mathbf{K}$:

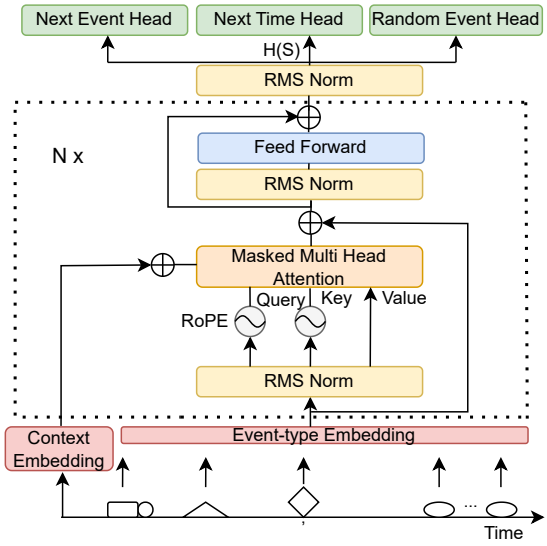

Figure 3: CarFormer architecture

$$\mathbf{Q} = \mathbf{R}^d_\Theta(\mathbf{W}^Q\mathbf{E} + \mathbf{CE}),$$
$$\mathbf{K} = \mathbf{R}^d_\Theta(\mathbf{W}^K\mathbf{E} + \mathbf{CE})$$

where $\Theta = \{\theta_i = \theta_0^{-2(i-1)/d}, i \in [1, 2, \ldots, d/2]\}, \theta_0 = 10^4$. More specifically, the inner product between query $\mathbf{q}_m$ and key $\mathbf{k}_n$ takes the event-type embedding $\mathbf{e}_m, \mathbf{e}_n$ where $m-n$ is their relative position with context $\mathbf{CE}_m$ and $\mathbf{CE}_n$:

$$\begin{aligned}\mathbf{q}_m^T\mathbf{k}_n &= (\mathbf{R}^d_{\Theta,m}(\mathbf{W}_q\mathbf{e}_m + \mathbf{CE}_m))^T\mathbf{R}^d_{\Theta,n}(\mathbf{W}_k\mathbf{e}_n + \mathbf{CE}_n)\\ &= \mathbf{e}_m^T\mathbf{W}_q\mathbf{R}^d_{\Theta,n-m}\mathbf{W}_k\mathbf{e}_n + \mathbf{e}_m^T\mathbf{W}_q\mathbf{R}^d_{\Theta,n-m}\mathbf{CE}_n +\\ &\quad \mathbf{CE}_m^T\mathbf{R}^d_{\Theta,n-m}\mathbf{W}_k\mathbf{e}_n + \mathbf{CE}_m^T\mathbf{R}^d_{\Theta,n-m}\mathbf{CE}_n\\ &= \text{(1): query-to-key} + \text{(2): query-to-ce}\\ &\quad \text{(3): ce-to-key} + \text{(4): ce-to-ce}\end{aligned} \tag{8}$$

where $\mathbf{R}^d_{\Theta,n-m} = (\mathbf{R}^d_{\Theta,m})^T\mathbf{R}^d_{\Theta,n}$ is a sparse orthogonal matrix. The additional terms (2), (3), (4) provide richer query and key representations when computing the attention scores. Thus modifying Equation 4:

$$\mathbf{A} = \text{softmax}\left(\frac{(\mathbf{R}^d_\Theta(\mathbf{W}^Q\mathbf{E} + \mathbf{CE}))(\mathbf{R}^d_\Theta(\mathbf{W}^K\mathbf{E} + \mathbf{CE}))^T}{\sqrt{3d}}\right)$$

Adding $\mathbf{CE}$ after the projection to query and key can be seen as a refinement of $\mathbf{Q}, \mathbf{K}$ by $\mathbf{T}, \mathbf{M}$, providing additional context to the attention scores. We also add a scaling factor of 3 to compensate for the additional terms in Equation 8.

Multi-Task Learning

Next Event Prediction. We use a standard language modeling objective which aims to minimize the cross-entropy loss between our output distribution $\hat{u}_i$ generated by our model's *Next Event Head* and the next event u_{i+1}. (Shou et al. 2024) used a BERT (Devlin et al. 2019) model trained on a masked event modelling task which is commonly used

for bidirectional models. However, they simultaneously applied a causal mask resulting in a loss of the bidirectional property. We argue that by doing so, we lose a lot of sample efficiency, thus we will stick to a standard next token prediction. $\hat{u}_i \in \mathbb{R}^V$ is the predicted probability distribution by the *Next Event Head*, which integrates an RMS normalization (Zhang and Sennrich 2019) and one linear layer. The cross-entropy loss between $\hat{u}_i$ and u_{i+1} (= a one-hot vector in $\{0,1\}^V$) is obtained by:

$$\mathcal{L}_c := -\sum_{i=0}^{L}\sum_{j=0}^{V} u_{i+1,j}\log(\hat{u}_{i,j})(1-\delta_{i,r}) \tag{9}$$

where $\delta_{i,r}$ is the Kronecker delta, which equals 1 when $i = r, r \in R$ (set of randomly generated events) and 0 otherwise.

Next Event Time Prediction. In addition, we compute the Huber loss (Jadon, Patil, and Jadon 2024) between the estimated inter-event time $\Delta\hat{t'}_i$ for the event u_i and the ground truth $\Delta t'_i = f_t(t_{i+1}, 10) - f_t(t_i, 10)$ to deal with outliers and prevent exploding gradients with $\beta = 1, \epsilon_i = \Delta\hat{t}'_i - \Delta t'_i$.

$$\mathcal{L}_t := \sum_{i=0}^{L}(1-\delta_{i,r})\begin{cases}0.5\epsilon_i^2 & \text{if } |\epsilon_i| < \beta,\\ (|\epsilon_i| - 0.5) & \text{otherwise,}\end{cases} \tag{10}$$

t' is obtained using a log, hence we are essentially computing a kind of Mean Squared Logarithmic Error (MSLE) but with a β, useful to stabilize training and help convergence.

Random Event Prediction. Finally, a binary classifier that predicts whether an event was true or randomly generated is added. We motivate this choice by several papers (Shou et al. 2024; Gao et al. 2020) stating that a model should learn when an event does not happen to reinforce the negative evidence of no observable events within each inter-event. At each step i, a random event is injected with probability p. If a random event is successfully injected, the process continues until a failure occurs (i.e., the event is not injected), this allows for multiple random events to be injected in a row, allowing for more complexity. The $\mathcal{L}_r$ loss is defined as the binary cross-entropy loss between the probability distribution $\hat{y}_i^r$ generated by our *Random Event Head* and the ground truth y_i^r at event step i:

$$\mathcal{L}_r := -\sum_{i=0}^{|R|} y_i^r \log(\hat{y}_i^r) + (1-y_i^r)\log(1-\hat{y}_i^r) \tag{11}$$

Total Loss. The total loss is defined as follows:

$$\mathcal{L} = \frac{1}{L-|R|}(\mathcal{L}_c + \alpha\mathcal{L}_t) + \beta\frac{1}{|R|}\mathcal{L}_r \tag{12}$$

where α, β are trade-off between the different loss, L the sequence length, R the set of random events injected in S.

5 EPredictor

Predicting only the next DTC in a sequence of DTC faults has its inherent limitation and remains a difficult task. (Hafeez, Alonso, and Riaz 2024) use DTCs those ECU, Base-DTC and Fault-Byte data have a cardinality of 83, 419

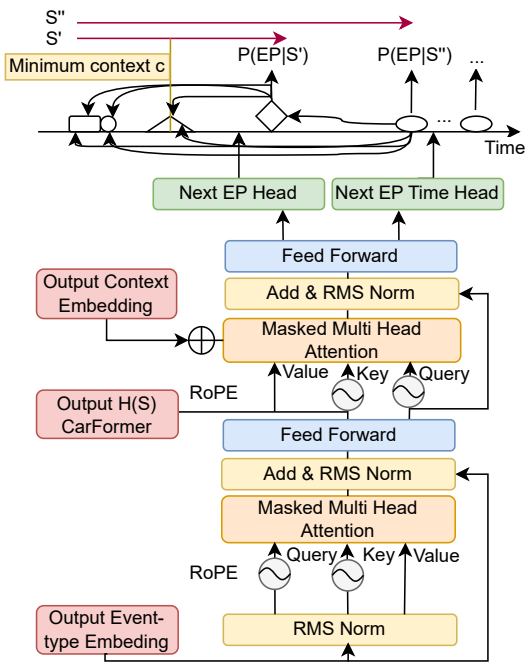

Figure 4: EPredictor architecture

and 64, respectively, and only report a 81% top-5 accuracy for next DTC prediction. This is because DTCs are not always correlated nor have causal links. Instead, we are also using repair and warranty data to predict more important events such as EPs (error patterns). Repair and warranty data differ from DTCs since they are manually defined by domain experts after observing all DTCs and characterize a whole sequence S and not an individual event u_i. Hence, we can define and say that a certain error pattern y has happened at index $i = L$ in a sequence S. Note, that multiple EPs can occur at the same time. We can now define a supervised multi-label classification learning problem of predicting EPs. With EPredictor, we leverage the *seq2seq* nature of Transformers, where CarFormer outputs a sequence $H(S)$ of tokens encoded in a high-dimensional space d, positioning tokens with similar characteristics nearby. Then, this hidden representation $H(S)$ is fed into EPredictor which acts as an autoregressive multi-label classifier for EPs. We approach EP prediction as a machine translation task. By utilizing the contextualized hidden states $H(S)$ from CarFormer as key and value (i.e., through *cross-attention*), this effectively transitions our model from a "seq2seq" to a "dtc2errorpattern" framework.

Multi-Label Event Prediction

Multi-label classification has recently gained interest in event prediction. (Zhang et al. 2020b) uses an LSTM for fault detection. More recently, (Shou et al. 2023) considers concurrent event predictions as a multi-label classification and models such data with a Transformer architecture.

To define the multi-label event prediction task with N labels (≡EPs), we reuse each $S = \{(u_i, t_i, m_i)\}_{i=0}^{L}$ and at-

tach a binary vector $\mathbf{y} \in [0,1]^N$ to indicate the EPs occurring at time t_L. It's important to note that $\mathbf{y}$ is invariant per sequence S, meaning for all events within S the ground truth $\mathbf{y}$ will be the same. By using a causal mask on both CarFormer and EPredictor, we enable predictive maintenance predictions since tokens can only attend to the previous ones, as shown in Figure 4. In our case, and in most real-world problems, EPs are highly imbalanced across our dataset, which is a considerably more challenging problem for our event prediction task, especially when the least occurring classes are the most important to detect (Zhang et al. 2020b). In natural language processing, traditional upsampling methods involve perturbation of S by shuffling and replacement of tokens. However, in event data, we cannot afford to lose spatial and temporal information. Therefore, we inject random events (u_i, m_i, t_i) with the same probability of $p = 0.05$ as in Section 4 and reuse the associated Algorithm. We up-sample the different EPs classes up to a minimum $\theta_1 = 6000$ and downsample the most popular ones down to a maximum $\theta_2 = 12000$. We drop also classes below 100 apparitions across the dataset. We define a minimum context $c = 30$ which acts as a "minimum history" of DTCs to retain. The *Next EP Head* output a vector of probabilities $\hat{y}_i = \text{sigmoid}(\text{MLP}_c(H_i))$ for each history $\{H_1, ..., H_i\}$, $i \in c, ..., L$ where $H_i \in \mathbb{R}^d$ is the generated hidden representation from EPredictor at step i. For the regression task we forecast the time till the EP(s) occurrence $\Delta\hat{t'}_i = \text{MLP}_t(H_i)$ where the ground truth is $\Delta t'_i = f_t(t_L, 30) - f_t(t_i, 30)$. Formally, we define our binary cross-entropy loss over the N possible EPs for one step i as follows:

$$\mathcal{L}_i^{ep} := -\frac{1}{N}\sum_{j=0}^{N} y_j \log(\hat{y}_{i,j}) + (1 - y_j)\log(1 - \hat{y}_{i,j}) \quad (13)$$

The total loss across S with L events and context c is:

$$\mathcal{L}^{ep} := \frac{1}{L-c}\sum_{i=c}^{L} \mathcal{L}_i^{ep} \quad (14)$$

We use the Huber loss (Jadon, Patil, and Jadon 2024) and define $\epsilon_i = \Delta\hat{t}'_i - \Delta t'_i$, with $\beta = 1$ thus:

$$\mathcal{L}^{t} := \frac{1}{L-c}\sum_{i=c}^{L} \begin{cases} 0.5\epsilon_i^2 & \text{if } \epsilon_i < \beta, \\ |\epsilon_i| - 0.5 & \text{otherwise,} \end{cases} \quad (15)$$

Our final loss to minimize is then : $\mathcal{L} = \mathcal{L}^{ep} + \gamma\mathcal{L}^{t}$

6 Experiments

We implemented and trained CarFormer and EPredictor models using PyTorch, the code is publicly available. Training details are included in the Appendix.

Ablation I: CarFormer Embeddings

Multiple CarFormer models with different choices of embeddings were evaluated on the next token prediction accuracy (ACC) and on the regression task with mean absolute percentage error (MAPE) and root mean square error (RMSE) to determine the best working CarFormer model. The different choices were *rot* (RoPE), *time* (absolute time embedding added to the input $\mathbf{U}$), *mileage* (also added to $\mathbf{U}$), *m2c*, and *c2m* are additional dot-products ((He et al. 2021)). In our case, we are more interested in the next event prediction task, thus we accept to lose some MAPE % over the ACC (%).

Model	ACC(%)	MAPE(%)	RMSE
rot-ce	**22.64**	3.2	0.04770
time	21.48	**2.9**	**0.04762**
time-mileage	21.38	3.0	0.04785
time-c2m-m2c	21.58	3.5	0.04794
time-m2c	21.52	3.6	0.04823
GPT	19.89	-	-

Table 3: Overall prediction performance of CarFormer with different embeddings. Best results are in bold.

Using only *time* gave the best MAPE (2.9) but not the best ACC (21.48), suggesting that other features might improve the model predictions. For the mileage integration, our intuition was that doing an early summation of the two embeddings $(\mathbf{T}, \mathbf{M})$ seemed to denature the input $\mathbf{U}$ (ACC of *time-mileage* $<$ ACC of *time*) but the mileage of the vehicle could help differentiate between different DTCs. We tried to modify the attention dot products which seemed to help the ACC a bit (*time-c2m-m2c*, *time-m2c*) but increased the MAPE drastically. So we fused it with $\mathbf{CE}$ directly in $\mathbf{Q}, \mathbf{K}$. Then, by applying a RoPE to the transformed input, the ACC increased while preserving the RMSE, leading to the best performing model, namely: *rot-ce*.

EPredictor Experiments

We used the micro-F1 score (Zhang and Zhou 2014) to assess the performance of the multi-label classification. To better understand and enhance our model's predictive maintenance capabilities, we introduce the concept of *Confident Predictive Maintenance Window (CPMW)* which represents the interval within which our model can make reliable predictive maintenance predictions (similar to the "prediction window" described in (Pirasteh et al. 2019)). We quantify this with the *CPMW Area Under Curve* (*CPMWAUC*). The F1 score, MAE and MAPE have been calculated on average for all observations in Table 4, and additionally for each history H_t^i in Figure 4 to understand how each model performs with different numbers of observations. To monitor the predictive maintenance capability of each model, the $\text{CPMWAUC}_{\text{f1}}$ and $\text{CPMWAUC}_{\text{mae}}$ were computed.

Ablation II: EPredictor Architecture & CPMW

We explored several architectural changes and their impact on the CPMW: we applied a RoPE (*rot*), a cross attention with *query* or *key* or *value* to the second multi-head attention block (*cross*), added the context embedding $\mathbf{CE}$ (*ce*) to layer *1* and/or *2*, applied a scaling factor of $\sqrt{3d}$ to $\mathbf{Q}, \mathbf{K}$ as shown in Equation 4 (*scale*), injected a relative matrix $\mathbf{S}_{rel}$ into the

Model	Micro F1 (%)	MAPE (%)	MAE	CPMWAUC$_{f1}$ ↑	CPMWAUC$_{mae}$ ↓
rotcross-query-key-ce-1-2	82.69	32.45	0.0268	52.80	0.888
rotcross-query-key-ce-2	82.69	**31.18**	0.0254	56.61	0.882
rotcross-query-ce-2	82.69	**31.18**	0.0254	53.00	0.884
rotcross-key-value-ce-2	**84.38**	33.47	0.0263	65.06	0.904
rotcross-key-value-scaled-ce-2	**84.38**	31.44	**0.0252**	**67.63**	**0.874**
rotnocross-ce-1-2	80.74	37.61	0.0275	42.95	0.927
cross-speed	83.53	34.73	0.0260	49.28	0.877
cross-mixffn	83.41	33.37	0.0270	54.39	0.891
time-cross-query	83.34	35.89	0.0275	45.71	0.896

Table 4: EPredictor evaluation results with different model architecture on the test set (no up- nor down-sampling)

attention scores (*speed*), and applied a mixed feed-forward network (*mixffn*) (Xie et al. 2021) to the mileage embedding. The *time* model refers to $\mathbf{T}$ which is also added to $\mathbf{E}$. Finally, we trained a model with and without the *Random Event Head*. Our experiments revealed several key insights: By applying a *cross* attention, we can see improvement in all metrics (*rotnocross-ce-1-2*), which is consistent with the machine translation analogy *"dtc2errorpattern"*. The best cross attention results were shown with $H(S)$ used as *key-value*. Adding the mileage via an MLP layer (*cross-mixffn*) seemed to help the MAPE (-2.5%), the CPMWAUC$_{\text{f1}}(+9)$ and also the CPMWAUC$_{\text{mae}}(-0.005)$ compared to *time-cross-query* model, suggesting that mileage is beneficial for both task. This makes sense since EPs are also dependent on the traveled distances between DTCs and the different stationary behavior of the vehicle. Furthermore, models incorporating a RoPE (*rot*) performed significantly better in both regression and classification tasks like in the pre-training, highlighting the performance of RoPE in machine translation tasks (Su et al. 2024). Adding $\mathbf{CE}$ to the last layer (*ce-2*) gave the best results, as opposed to adding it to both layers (*ce-1-2*). Surprisingly, doing feature engineering on $\mathbf{T}, \mathbf{M}$ with a *speed* matrix $\mathbf{S}_{\text{rel}}$ didn't help the metrics, which could indicate some missing modalities (e.g. mileage) during training, thus $\mathbf{CE}$ is more adapted for real-world scenarios. We monitored the need for our *Random Event Head* and noticed $+1.2\%$ in the F1 Score and $+10.1$ in the CPMW$_{f1}$. When taking the best performing model (*rotcross-key-value-scaled-ce-2*), the model entered the CPMW after 81 observations i.e. **half of the sequence**. Within this window, the model obtained an error of $\approx \mathbf{65} \pm \mathbf{14h}$ when estimating the time of EP occurrence (Figure 6), highlighting the model's predictive capability within the CPMW. Otherwise, the average absolute error across all observations was approximately 58.4 ± 13.2h. By experimenting with these modifications, we aimed to identify the optimal architecture for predictive maintenance. The findings reveal that cross attention, context embedding in the second layer, and scaled attention significantly improve performance within the CPMW.

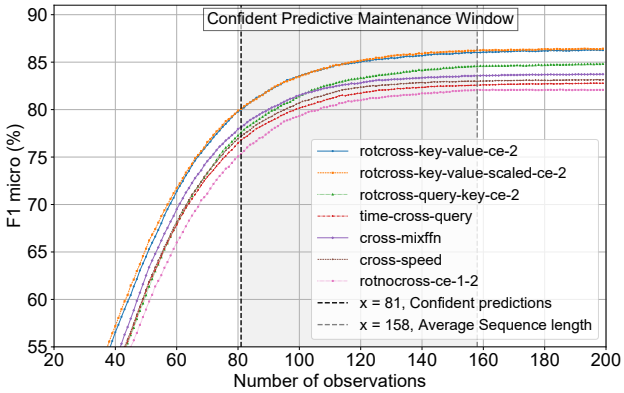

Figure 5: F1 Score comparison with multiple Epredictor architectures in function of the number of observations.

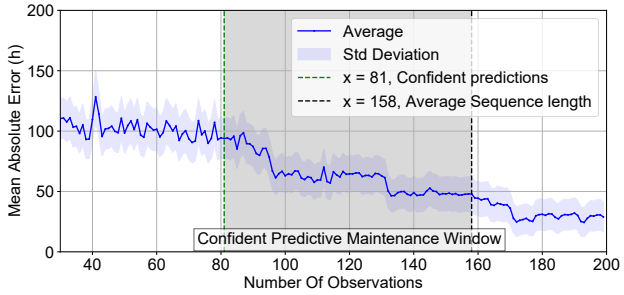

Figure 6: Evolution of the MAE in function of the number of observations for the best performing model.

7 Conclusion

This study bridges the gap between traditional event sequence modeling and fault event prediction in vehicles by using two language models. We have demonstrated through specific relevant metrics that our approach can accurately perform predictive maintenance by effectively predicting *when* and *what* error patterns are likely to occur, even with continuous, unbalanced, and high cardinality data. In real-world settings, EPredictor is easy to use in a car. After each DTC occurrence and until reaching a minimum number of observations is reached, we would then infer the most likely EP and its time of occurrence. If the model is confident enough, the user will be alerted to an impending critical fault and directed to a nearby dealer, hence enhancing vehicle safety on the road and reducing maintenance costs.

References

Bachmann, G.; and Nagarajan, V. 2024. The Pitfalls of Next-Token Prediction. In *ICLR 2024 Workshop: How Far Are We From AGI*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1877–1901. Curran Associates, Inc.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *North American Chapter of the Association for Computational Linguistics*.

Du, N.; Dai, H.; Trivedi, R.; Upadhyay, U.; Gomez-Rodriguez, M.; and Song, L. 2016. Recurrent Marked Temporal Point Processes: Embedding Event History to Vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 1555–1564. New York, NY, USA: Association for Computing Machinery. ISBN 9781450342322.

Gao, T.; Subramanian, D.; Shanmugam, K.; Bhattacharjya, D.; and Mattei, N. 2020. A Multi-Channel Neural Graphical Event Model with Negative Evidence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34: 3946–3953.

Hafeez, A. B.; Alonso, E.; and Riaz, A. 2024. DTC-TranGru: Improving the performance of the next-DTC Prediction Model with Transformer and GRU. *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*.

Hafeez, A. B.; Alonso, E.; and Ter-Sarkisov, A. 2021. Towards Sequential Multivariate Fault Prediction for Vehicular Predictive Maintenance. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 1016–1021.

Hawkes, A. G. 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1): 83–90.

He, P.; Liu, X.; Gao, J.; and Chen, W. 2021. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In *International Conference on Learning Representations*.

Jadon, A.; Patil, A.; and Jadon, S. 2024. A Comprehensive Survey of Regression-Based Loss Functions for Time Series Forecasting. In Sharma, N.; Goje, A. C.; Chakrabarti, A.; and Bruckstein, A. M., eds., *Data Management, Analytics and Innovation*, 117–147. Singapore: Springer Nature Singapore. ISBN 978-981-97-3245-6.

Lin, H.; Wu, L.; Zhao, G.; Pai, L.; and Li, S. Z. 2022. Exploring Generative Neural Temporal Point Process. *Transactions on Machine Learning Research*.

Pirasteh, P.; Nowaczyk, S.; Pashami, S.; Löwenadler, M.; Thunberg, K.; Ydreskog, H.; and Berck, P. 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*, WIDM'19. New York, NY, USA: Association for Computing Machinery. ISBN 9781450362962.

Shaw, P.; Uszkoreit, J.; and Vaswani, A. 2018. Self-Attention with Relative Position Representations. In Walker, M.; Ji, H.; and Stent, A., eds., *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 464–468. New Orleans, Louisiana: Association for Computational Linguistics.

Shchur, O.; Türkmen, A. C.; Januschowski, T.; and Günnemann, S. 2021. Neural Temporal Point Processes: A Review. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 4585–4593. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Shou, X.; Gao, T.; Subramanian, D.; Bhattacharjya, D.; and Bennett, K. P. 2023. Concurrent Multi-Label Prediction in Event Streams. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8): 9820–9828.

Shou, X.; Subramanian, D.; Bhattacharjya, D.; Gao, T.; and Bennet, K. P. 2024. Self-Supervised Contrastive Pre-Training for Multivariate Point Processes. *ArXiv*, abs/2402.00987.

Su, J.; Ahmed, M.; Lu, Y.; Pan, S.; Bo, W.; and Liu, Y. 2024. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomput.*, 568(C).

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Xie, E.; Wang, W.; Yu, Z.; Anandkumar, A.; Alvarez, J. M.; and Luo, P. 2021. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 12077–12090. Curran Associates, Inc.

Zhang, B.; and Sennrich, R. 2019. Root Mean Square Layer Normalization. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Zhang, M.-L.; and Zhou, Z.-H. 2014. A Review On Multi-Label Learning Algorithms. *Knowledge and Data Engineering, IEEE Transactions on*, 26: 1819–1837.

Zhang, Q.; Lipani, A.; Kirnap, O.; and Yilmaz, E. 2020a. Self-Attentive Hawkes Process. In III, H. D.; and Singh, A., eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 11183–11193. PMLR.

Zhang, W.; Jha, D. K.; Laftchiev, E.; and Nikovski, D. 2020b. Multi-label Prediction in Time Series Data using Deep Neural Networks. *CoRR*, abs/2001.10098.

Zuo, S.; Jiang, H.; Li, Z.; Zhao, T.; and Zha, H. 2020. Transformer Hawkes Process. In III, H. D.; and Singh, A., eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 11692–11702. PMLR.