Predictive Maintenance of Vehicles in Connected Environment

Abdul Basit Hafeez
School of Science and Technology (SST)
Department of Computer Science
City St George's, University of London

A thesis submitted in partial fulfilment of the requirement for the degree of Doctor of Philosophy

July 18, 2025

To my wife, whose love and support have been a constant source of strength during the highs and lows of my PhD.

To my parents and sisters, for helping me to be who I am, and for their support and prayers.

To my son, Abdullah, whose love and laughter have fueled my determination to complete this journey.

To those who supported, motivated, and prayed for my success.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 65,000 words including appendices, bibliography, footnotes, tables and equations and has less than 150 figures. I grant powers of discretion to the City St George's, University of London librarian to allow the dissertation to be copied in whole or in part without further reference to myself (the author). This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgment.

Abdul Basit Hafeez

July 18, 2025

Acknowledgements

I am profoundly grateful to Allah Almighty for His endless blessings, for providing me with the opportunity to pursue this PhD, and for giving me the strength to complete my dissertation on time. I sincerely appreciate my supervisors, Eduardo Alonso and Atif Riaz, for allowing me the privilege of researching a field I am passionate about and for their unwavering guidance throughout my academic journey. I truly feel fortunate to have had supervisors who were consistently approachable, supportive, and generous with their time and expertise. Their mentorship has inspired me, and I aspire to embody their values in my future endeavours.

I would also like to thank my industrial supervisors from Bosch, Hammouchene Rachid and Ansell Keith, for their constant support and guidance throughout the project. Their assistance during these years has been invaluable, and this research would not have been possible without the sponsorship from Bosch.

Moreover, I am deeply grateful to my examiners, Dr. Marin Lujak and Dr. Juan Guzman, for their dedication and interest in my work. Their constructive feedback significantly enhanced the quality of this dissertation. I also offer my sincere appreciation to Esther Mondragón, Senior Tutor for Research, for graciously chairing my viva and contributing to this pivotal milestone in my academic journey.

Lastly, I want to thank everyone who supported me during my PhD, including Ann Marie from the postgraduate research course office, all my friends, and fellow PhD researchers at CITAI, especially Esther Mulwa, for their availability and support throughout my PhD journey.

Abstract

Predictive maintenance represents a data-driven methodology that applies machine learning and analytics to foresee equipment failures before their occurrence, thus reducing downtime and associated maintenance costs. Historically, the practice of predictive maintenance, especially in the context of vehicles, has relied on anomaly detection techniques applied to sensor data. In recent developments, vehicular predictive maintenance has transitioned from leveraging raw sensor inputs directly to employing fault events recorded within On-Board Diagnostic systems (OBDs). Rather than delivering raw sensor data, OBDs provide drivers and technicians with diagnostic information derived from various Electronic Control Units (ECUs) in vehicles, typically represented as Diagnostic Trouble Codes (DTCs).

Despite their significance and widespread use in the automotive industry, the application of conventional machine learning techniques to predictive maintenance using DTCs presents considerable challenges: DTCs are non-numeric, the spectrum of DTC codes is exceedingly vast, and they can coexist with additional attributes such as fault-bytes and Electronic Control Units (ECUs). These challenges have prompted researchers to examine a limited subset of DTCs at one time and to employ basic algorithms. Moreover, most algorithms applied to DTCs are heavily reliant on repair and warranty data to formulate problems within a supervised learning paradigm, such as categorizing sequences of events as faulty or non-faulty. However, in the absence of access to such data or clear indicators of vehicle non-operability, implementing supervised learning techniques, which necessitate substantial quantities of labeled data, can be difficult, if not infeasible.

This study initially re-conceptualized the task of vehicle fault prediction as a self-supervised next-DTC prediction problem, introducing a novel architecture that harnesses the strengths of deep learning models to directly confront the intrinsic

complexity of DTCs. This is achieved by learning dense representations of DTC events through the use of neural embeddings, applied separately to each DTC attribute. Such a method enables our models to utilize sequential algorithms such as LSTM layers, thereby facilitating precise predictions of the subsequent event with consideration for all three attributes per timestep.

The second approach proposes an architecture that combines Gated Recurrent Units (GRUs) with an attention mechanism to succinctly encapsulate the complete DTC sequence into low-dimensional embeddings, facilitating efficient representation of multivariate event sequences, thereby enhancing accuracy, interpretability, and the capability for semantic search of individual DTCs and their associated sequences.

The third approach consolidates the advantages of transformer and GRU models, achieving a superiority of approximately 2% in the top-5 accuracy benchmark for the next-DTC prediction task. This model illustrates that large models while achieving outstanding performance in state-of-the-art research, do not necessarily operate optimally in domains constrained by data size.

Finally, we developed an improved variant of the DTC-TranGru model, referred to as DTC-GOAT, which incorporates various optimization techniques to augment prediction accuracy. Furthermore, we demonstrated how the ensemble approach, which entails the combination of multiple models for next-DTC prediction, can enhance top-5 accuracy outcomes compared to individual models.

Contents

Acronyms

ABS Anti-Lock Braking System.

ANN Approximate Nearest Neighbors.

DTC Diagnostic Trouble Codes.

ECU Electronic Control Unit.

EDA Exploratory Data Analysis.

EMB Embedding.

EOS End of Sequence.

FFN Feed-Forward Network.

GRU Gated Recurrent Unit.

IoT Internet of Things.

LSTM Long Short Term Memory.

LSTM(s) Long Short Term Memory Network(s).

OBD On-Board Diagnostic systems.

OHE One-Hot Encoding.

RNN Recurrent Neural Network.

SMFP Sequential Multivariate Fault Prediction.

TCM TELEMATIC CONTROL MODULE.

TM Transmission Module.

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation examines the utilization of sequential algorithms and neural embeddings to enhance the representation of Diagnostic-Trouble Codes (DTC) and to leverage them for predictive maintenance within an end-to-end self-supervised framework. This chapter commences by sharing the motivation behind employing DTCs for predictive maintenance and the necessity for a self-supervised methodology. Subsequently, the contributions of the dissertation are delineated, followed by a compilation of papers published as part of this research.

1.1 Motivation

Industrial systems, ranging from small machines to vehicles, rely on maintenance for their durability. In recent years, instead of fixing the machines after a fault occurs, companies have begun to invest in various techniques that can prevent faults in advance. The most popular approach to system maintenance, until recently, relied on *Preventive Maintenance* [6]. Preventive maintenance involves scheduled, proactive, and routine inspections of the system that help reduce faults and prevent them before they arise. It seeks to minimise major breakdowns and failures by addressing small problems ahead of time.

Conversely, predictive maintenance *Predictive Maintenance* [67, 86] employs machine learning algorithms to anticipate maintenance requirements and system failures by scrutinizing historical data. This involves examining correlations and uncovering complex patterns in data related to system failures, as well as activities undertaken for diagnosis and repair. Preventive and corrective maintenance necessitates the engagement of labour and resource allocation for periodic maintenance scheduling. In contrast, predictive maintenance facilitates cost reduction [101], diminishes equipment downtime [1], and extends the longevity of components [60]. Owing to these advantages, predictive maintenance is being increasingly integrated across various industries [78, 4, 68, 90].

More recently, advancements in industrial machines, especially vehicles, have led to the development of embedded diagnostic systems called Electronic Control Modules (ECUs) [98], which produce diagnostic fault codes. These fault events, known as Diagnostic Trouble Codes (DTC) [85], have begun to dominate traditional predictive maintenance approaches, shifting the focus from sensory data [52] collected by numerous sensors in these machines to events generated by the diagnostic modules. Since fault events are non-numeric, they are difficult to use directly with machine learning algorithms and require some form of numeric representation [83]. Obtaining this representation is challenging due to the multiple attributes, each with a high number of unique classes.

1.2 Role of DTCs in Predictive Maintenance

Diagnostic modules embedded within modern vehicles play a crucial role in monitoring the health and performance of various automotive systems. These modules continuously collect and analyze data related to vehicle operations, generating Diagnostic Trouble Codes (DTCs) when anomalies or faults are detected. The systematic use of these DTCs enables a proactive approach to vehicle maintenance, commonly referred

to as predictive maintenance. By analyzing DTCs and their patterns, maintenance teams can forecast component degradation and schedule repairs or replacements at optimal times, thus avoiding unexpected failures. This approach contrasts with traditional reactive maintenance, which typically occurs only after a fault has disrupted vehicle function.

Predictive maintenance driven by diagnostic data offers substantial advantages in terms of cost efficiency, safety, and overall system reliability. Early fault detection minimizes the likelihood of severe failures, which can lead to expensive repairs and extended downtime. Timely interventions not only extend the lifespan of critical components but also prevent secondary damage that may arise from neglected issues. From a safety standpoint, predictive maintenance ensures the continuous functionality of essential vehicle systems such as braking, steering, and engine control, thereby reducing the risk of accidents linked to sudden malfunctions. The consistent performance and reliability achieved through this method also contribute positively to customer satisfaction and brand reputation, particularly in high-end automotive markets.

The integration of diagnostic systems and DTC analysis thus forms the foundation of modern predictive maintenance strategies, enabling data-driven decision-making that improves vehicle safety, reduces operational costs, and enhances long-term performance.

1.3 Aim and Objective

The primary aim of this dissertation is to explore, evaluate, and develop end-to-end self-supervised prediction algorithms [32], which can use these DTC events to facilitate the identification of maintenance needs for vehicles in connected environments. Given the limitations in current research and to achieve the broader objective, we intend to explore the following goals.

1. In the absence of warranty and repair data, when it is not feasible to frame the problem using a supervised approach [40], how can Diagnostic Trouble Code (DTC) fault events be effectively utilized for predictive maintenance?

2. Does the high cardinality of multi-attribute DTC events constrain the implementation of advanced end-to-end machine learning algorithms for the identification of potential issues in vehicles? This inquiry is explored in Chapter 4.

3. Can enhanced representations for DTC events and their associated attributes be proposed and developed? How does this new representation facilitate the application of machine learning and deep learning algorithms? This inquiry is examined in Chapter 4.

4. Does the resulting new representation enable the reformulation of the problem into a more comprehensive end-to-end self-supervised next-DTC prediction problem, rather than focusing solely on a collection of DTC events? This inquiry is covered in Chapter 4.

5. Is it feasible to employ the developed algorithm in contexts where decision explanations are essential, and can the outcomes of the models be practically interpreted by field engineers? This inquiry is addressed in Chapter 5.

6. Can we integrate the approach to predict and retrieve similar DTC sequences and individual DTC events, or is there a necessity for specialized models tailored to different use cases? This inquiry is addressed in Chapter 5.

7. Considering the advancements of sequential prediction algorithms such as transformers, can a larger model simply replace a smaller sequential model to enhance the model's accuracy? This inquiry is addressed in Chapters 6 and 7.

1.4 Contributions

In this dissertation, we introduce a self-supervised learning methodology aimed at predicting the subsequent multivariate Diagnostic Trouble Code (DTC) event within a sequence thereof. This includes the development of various architectural frameworks employing sequential prediction algorithms to enhance DTC event prediction accuracy. Our research further encompasses an improved representation of DTC events and the implementation of sequential algorithms to predict the forthcoming DTC event, encompassing all three attributes. .

The following constraints pertain specifically to the objects and questions delineated in the preceding section:

- In instances where warranty and repair data, essential for supervised learning, are unavailable, we devised a self-supervised prediction methodology for subsequent DTC events. This approach incorporates dense DTC event representations via neural embeddings and the application of algorithms adept at managing sequential dependencies, such as Recurrent Neural Networks (RNNs).

- To facilitate model interpretability and expedite semantic retrieval of DTC sequences, we designed a dense DTC encoder architecture that yields a low-dimensional representation of the DTC sequence. This compact representation can be employed in various downstream applications, including retrieval tasks and semantic searches.

- We propose a hybrid model integrating a lightweight transformer network with Gated Recurrent Units (GRU) to enhance performance in next-DTC prediction tasks, particularly in scenarios involving limited data, where deploying standalone large models is infeasible.

- We also present an optimized architecture of the Transformer-GRU model that enhances information flow and interconnectedness, thereby improving next-

DTC prediction efficacy. Additionally, we propose a straightforward ensemble method to combine diverse architectures, thereby augmenting the performance of the next-DTC prediction task.

1.5 List of Publications

As a part of this dissertation work, number of articles have been published in peer-reviewed conferences. Part of this dissertation is based of these

- A. B. Hafeez, E. Alonso and A. Ter-Sarkisov, "Towards Sequential Multivariate Fault Prediction for Vehicular Predictive Maintenance," 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), Pasadena, CA, USA, 2021, pp. 1016-1021, doi: 10.1109/ICMLA52953.2021.00167,
- A. B. Hafeez, E. Alonso and A. Riaz, "DTCEncoder: A Swiss Army Knife Architecture for DTC Exploration, Prediction, Search and Model Interpretation," 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA), Nassau, Bahamas, 2022, pp. 519-524, doi: 10.1109/ICMLA55696.2022.00085.
- Abdul Basit Hafeez, Eduardo Alonso, and Atif Riaz. 2024. DTC-TranGru: Improving the performance of the next-DTC Prediction Model with Transformer and GRU. In Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24). Association for Computing Machinery, New York, NY, USA, 927–934. https://doi.org/10.1145/3605098.3635962

1.6 Dissertation Outline

The structure of this dissertation is as follows: Chapter 2 provides background information on PhD, Diagnostic Modules, and Diagnostic Trouble Codes. Chapter

3 offers a comprehensive literature review on sensor-based and DTC-based predictive maintenance research. Chapter 4 discusses the development and implementation of new representations for a novel approach to next-DTC prediction using end-to-end self-supervised learning. In this chapter, we explain how these representations can effectively handle high-cardinality features and sequential dependencies between events, and can subsequently be fed into the proposed next-DTC prediction model, which can potentially overcome the limitations of traditional supervised learning approaches. Chapter 5 highlights how the algorithm developed for the next-DTC prediction problem can enable field engineers to understand the reasoning behind the predictions and facilitate exploration tasks, such as retrieving similar DTC sequences or individual DTC events. This chapter introduces an architecture that learns dense representations of DTC sequences using the Attention mechanism and demonstrates how these representations can be utilized for other use cases. In Chapter 6, we present a hybrid architecture that combines Transformer and Gated Recurrent Unit (GRU) architectures. Chapter 7 proposes a structural improvement of the model utilizing a Transformer and GRU, aiming to enhance next-DTC prediction accuracy further. In this chapter, we also suggest combining different models to advance the accuracy of the DTC prediction task even more. Chapter 8 concludes the dissertation by summarizing the key findings, discussing implications for the automotive industry, and outlining avenues for future research.

Chapter 2

Background

One of the sponsors of this PhD programme is Bosch Automotive, specifically the Diagnostic Software Engineering Solutions Design team (now EPAS). As an industrial PhD sponsored by Bosch Automotive, the scope and focus of this PhD were primarily on the use of Diagnostic Trouble Codes for predictive maintenance in vehicles. It was anticipated that by the end of the PhD, we would be obtaining data from the vehicles in real-time, rather than data collected in the workshop; however, there were some unexpected delays due to COVID and Bosch's agreement with the customer.

A key limitation of the dataset used in this research is the absence of warranty claims data, which is typically a crucial component in predictive maintenance applications. This scarcity of rich contextual information necessitated a reframing of the problem, shifting focus towards the analysis of Diagnostic Trouble Codes (DTCs) themselves as the main source of insight for predicting vehicle faults. By examining the patterns and relationships within DTC sequences, researchers can identify potential issues before they escalate into major failures, thereby facilitating proactive maintenance strategies that mitigate costs associated with downtime and unforeseen repairs. This approach acknowledges the unique characteristics of the dataset while leveraging the inherent value of the DTC data to drive predictive insights in connected vehicles.

The subsequent sections provide foundational insights into diagnostic modules, the application of Diagnostic Trouble Codes for predictive maintenance, connected vehicles, and the dataset composition. To emphasize the self-supervised learning paradigm utilized in the problem formulation of this dissertation within a machine learning framework, this chapter culminates with an overview of machine learning and its various categories, including supervised, unsupervised, and self-supervised learning.

2.1 Fault Events-Based Predictive Maintenance

Many industrial machines, including connected cars, feature troubleshooting modules that generate fault events, commonly referred to as Diagnostic Trouble Codes (DTCs). In some cases, access to raw sensory data is severely limited or entirely absent, which restricts traditional approaches to analyzing event data. This limitation has led to the necessity for algorithms that can be applied to event-based systems.

The dataset utilised in this work is provided by Bosch Automotive Service Solutions. Diagnostic data, stored in the memory of electronic modules and control units, is collected either during a diagnostic session or streamed to the cloud periodically in connected cars. When a car visits the workshop, these DTCs assist engineers in identifying the causes of problems and determining if any module is malfunctioning. As will be discussed in section 2.5, the abundance of attributes in the data often makes it challenging for engineers to ascertain whether the current pattern will affect other modules in the future, thus hindering them from taking proactive preventive measures to save costs. By leveraging historical patterns, classifying a pattern within the failure category, or predicting the next probable failure can aid engineers in preventing major failures, which can be deemed the essence of event-based predictive maintenance.

2.2 Fault Detection in Connected Cars

In connected cars, there is a mechanism of bidirectional communication between the systems inside the vehicle and the outside world, for example, cloud servers and edge devices. This is enabled by the access to the internet and the ability to transfer the data to make the communication effective.

Having the ability to communicate with other systems efficiently, companies have started to log the data (sensor, fault events, other vehicle information) within the car and even transfer it in the form of data streams to external systems, i.e., cloud servers. This mechanism of connected communication can help in leveraging machine learning and data analytic-based solutions for reducing fault counts and improving the vehicle maintenance system through intelligent decisions.

2.3 On-Board Diagnostic and ECUs

OBDs (On-Board Diagnostics) and ECUs (Electronic Control Units) are essential components in modern vehicles. The OBD system acts as a vehicle's self-diagnostic and reporting system, allowing for the monitoring and analysis of various functions, including emissions control and engine performance. It gathers data from several sensors and transmits it to the ECU, a computer responsible for managing specific functions within the vehicle, such as the engine, transmission, brakes, and other vital systems. The ECU processes the information from the OBD system, making real-time adjustments to enhance the vehicle's performance and efficiency.

2.4 Diagnostic Trouble Codes

Diagnostic Trouble Codes (DTCs) are alphanumeric codes generated by the OBD system when it detects an issue with the vehicle's systems or components. Each code

corresponds to a specific fault or malfunction, making it easier for technicians to identify the problem. DTCs typically consist of five characters, with the first character indicating the system affected (e.g., "P" for powertrain issues, "C" for chassis), followed by numbers that pinpoint the exact nature of the problem. Mechanics use these codes to diagnose and troubleshoot issues, ensuring efficient and accurate repairs.

2.5 Dataset Structure

Each observation in a dataset is an error code denoting a fault that occurred in some module of the car. It has some attributes related to the car and the session where this event was recorded. Examples of such attributes include the number of miles that the vehicle has travelled and the time of the session. These attributes also help in sorting these events by time and mileage. A few important attributes of the dataset are mentioned below:

- Vin-id: ID of the vehicle.
- Mileage: Total distance travelled by the vehicle.
- DTC-ID: Id of the module generating the error log.
- Ecu (Module): Main module where the fault has occurred.
- Base-dtc: Sub-module where the fault has occurred.
- Fault-byte: Specific chip or location of the fault within sub-module.
- Session-datetime: Date and time of the session
- DTC-Status: Pending/Confirmed etc
- DDOO: It is the field that reflects the time passed since the record of error and session (when data is captured)

The dataset comprises diagnostic trouble code (DTC) sequences collected from Jaguar Land Rover (JLR) workshops, with data provided and supported through collaboration with Bosch, a key automotive supplier and research partner. Bosch, as a leading provider of automotive electronics and diagnostic solutions, supplies critical components such as electronic control units (ECUs) and diagnostic tools that enable the collection and analysis of vehicle fault data. This collaboration facilitates access to high-quality, real-world diagnostic datasets essential for advanced fault analysis and reliability studies.

The initial dataset consisted of over 450,000 sequences obtained from a larger pool of raw workshop data. A rigorous filtering process was applied to enhance data quality and relevance. Specifically, sequences containing fewer than five events were discarded due to insufficient length for temporal analysis. Immediate duplicate events within sequences were removed to prevent redundancy, and DTC events missing both timestamp and mileage information were excluded to maintain data integrity. After the preprocessing and filtering, we were left with 250,000 sequences of DTCs, each coming from a unique vehicle.

Following preprocessing, the dataset includes 486 unique base DTCs distributed across 83 ECU classes and 64 fault byte identifiers. The number of classes and identifiers retained reflects the filtering criteria rather than manual selection. Sequence lengths vary, representing diverse vehicle conditions and diagnostic histories across multiple JLR car models.

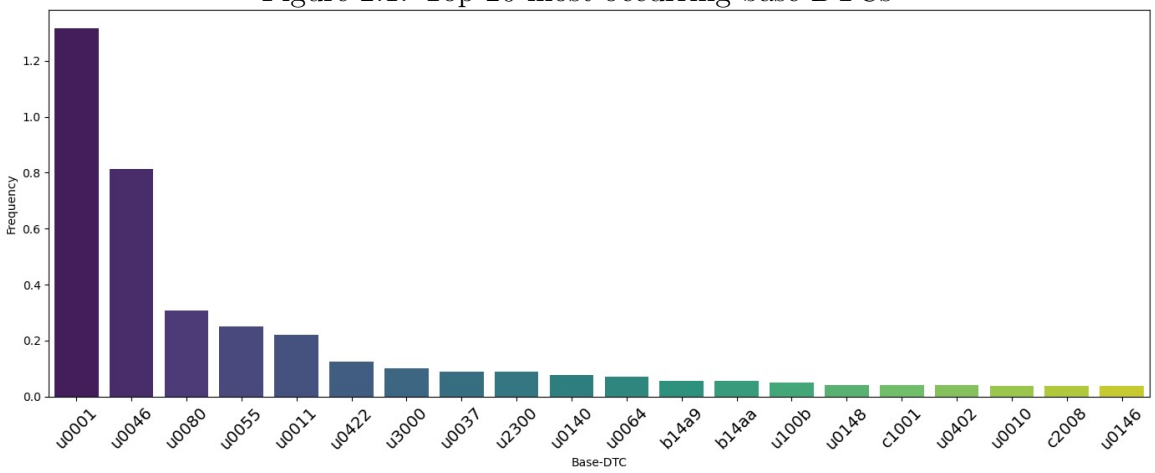
2.5.1 Common classes and frequencies

The top 20 most frequently occurring base DTCs in the dataset include codes such as U0001, U0046, and U0080. The code U0001 corresponds to issues on the high-speed Controller Area Network (CAN) communication bus, indicating general communication failures within this critical vehicle network. U0046 signifies a communication problem specifically within the "Vehicle Communication Bus C," which is a sub-

network responsible for data exchange between certain control modules. U0080 reflects a loss of communication on the CAN bus, representing interruptions in the data flow between various vehicle control units. These communication-related DTCs are among the most prevalent faults recorded, highlighting the importance of network reliability in vehicle diagnostics.

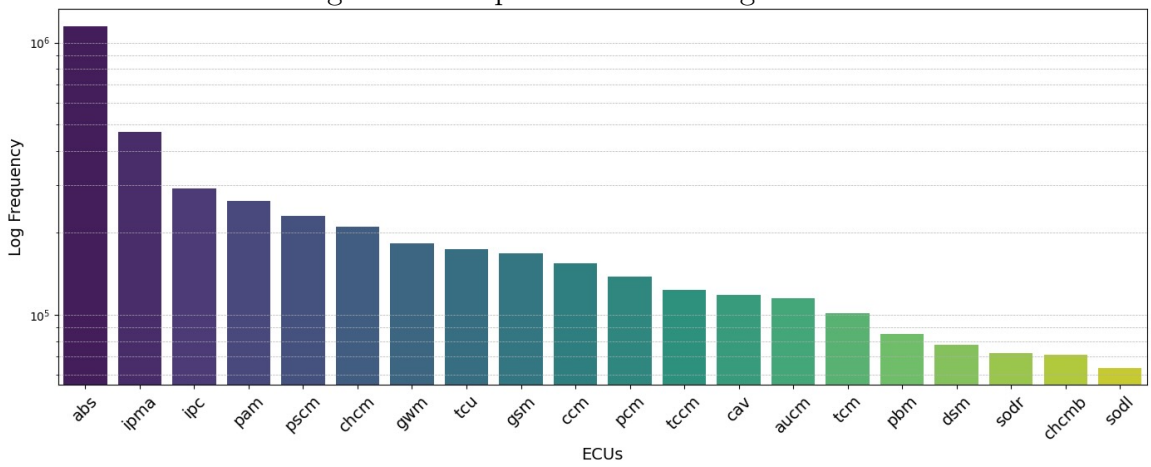
Since the dataset contains 468 unique base DTC classes, only the top 20 most frequent base DTCs are presented for clarity. Figure 2.5.1 shows the frequency distribution of these top 20 base DTCs, highlighting the predominance of communication bus-related faults, alongside other significant fault codes such as U0055 and U0011.

Figure 2.1: Top 20 most occurring base DTCs



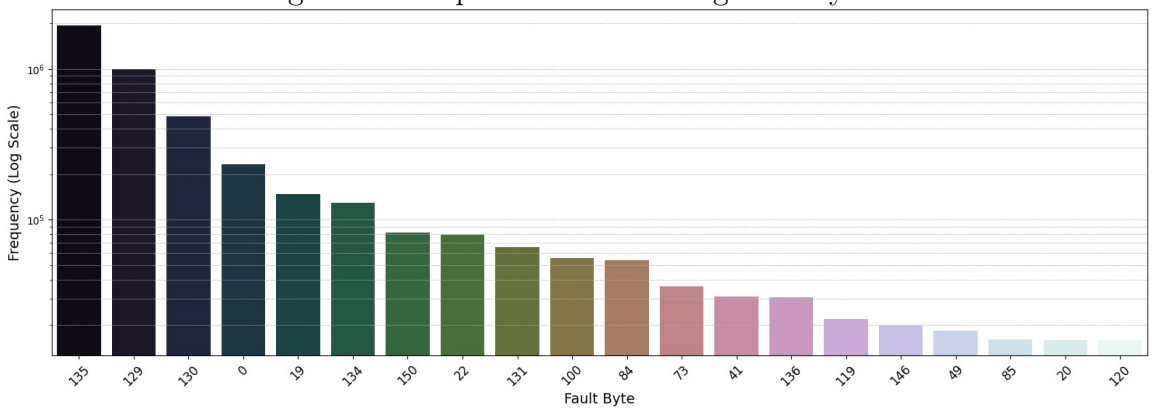
The 20 most frequently occurring ECUs are showing in figure 2.5.1, each identified by a standardized module name and its corresponding occurrence count. The most prominent entries include the Anti-lock Braking System (ABS), Image Processing Module A (IPMA), Instrument Panel Cluster (IPC), Parking Aid Module (PAM), and Power Steering Control Module (PSCM)

Figure 2.2: Top 20 most occuring ECUs



The fault bytes represent standardized diagnostic identifiers that encode specific failure modes or error types detected by vehicle ECUs. As shown in figure **??**, the most common values include '135', '129', and '130'

Figure 2.3: Top 20 most occurring fault-bytes



2.6 Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) that concentrates on creating algorithms that learn patterns from data and improve their predictive capabilities autonomously. While AI covers a wide array of methods designed to mimic human intelligence, ML lays the mathematical and statistical groundwork allowing

systems to identify patterns, make informed decisions, and predict future occurrences based on previous data.

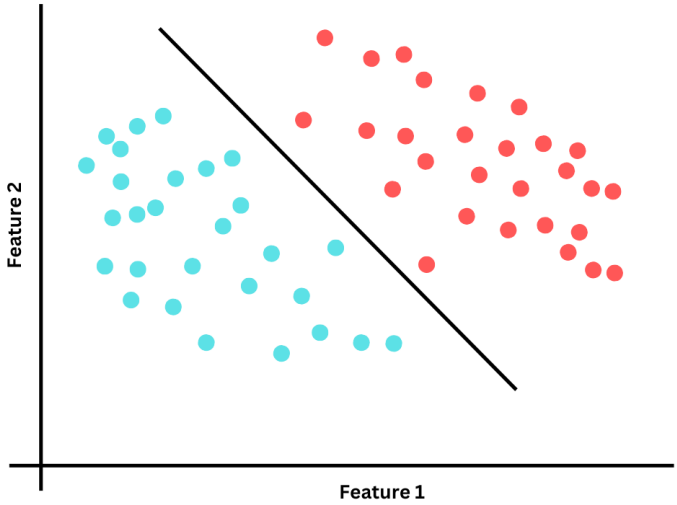
In predictive maintenance, ML is crucial for foreseeing equipment failures, optimizing maintenance schedules, and reducing downtime by evaluating sensor data, operational logs, and past maintenance records.

2.6.1 Supervised Learning

Supervised learning is a paradigm within machine learning wherein the model is trained utilizing a labeled dataset, denoted as (X, Y), where X signifies the input features and Y corresponds to the associated target labels. The algorithm's objective is to ascertain a mapping function $f : X \to Y$ that attenuates the deviation between predicted and actual outputs. Illustrations of supervised learning comprise:

- **Classification:** Classification is the task of predicting discrete categories or classes based on input data. The model learns from labeled examples to classify new instances into predefined classes.

Figure 2.4: Example of classification with linear decision boundary

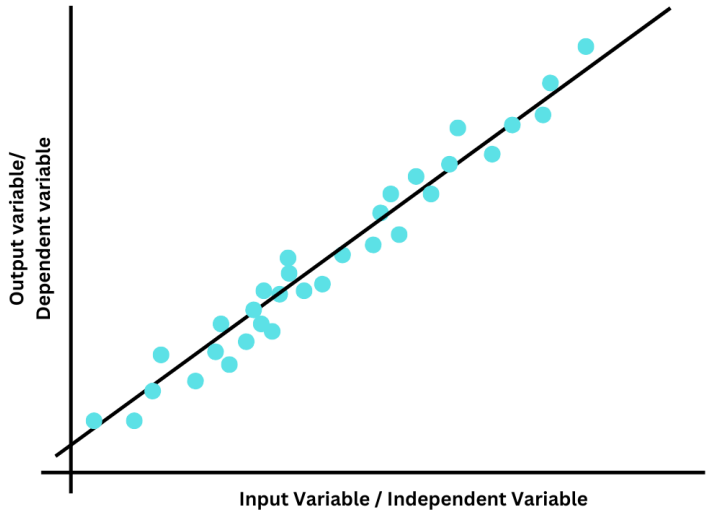


For example, a binary classification model predicts whether a machine will fail

within a specified timeframe (e.g., "fail" or "no fail") using features such as vibration levels, temperature, and pressure.

- **Regression:** Regression involves predicting continuous numerical values based on input data. The dataset is represented as (x, y), where x refers to the dependent or input variables and y denotes the independent or output variable. The model learns the relationship between these input variables and continuous outcomes.

Figure 2.5: Simple example of fitting a Linear model for Regression task, here dependent or input variable is on X-axis while output or Independent variable is on Y-axis



Example of regression in prediction maintenance will include a regression model estimating the remaining useful life (RUL) of a component based on historical sensor data.

2.6.2 Unsupervised Learning

Unsupervised learning entails training models on datasets devoid of labeled outputs, with the aspiration of identifying latent patterns, categorizing analogous data points,

or detecting anomalies. The dataset is represented as X, where the lack of labels differentiates it from supervised learning

Categories for unsupervised learning include the following.

- **Clustering:** Clustering is the process of grouping data points into clusters based on their similarities, without prior knowledge of the categories. For instance, techniques such as k-means clustering can group machines exhibiting similar operational patterns, facilitating the identification of anomalous equipment behavior.

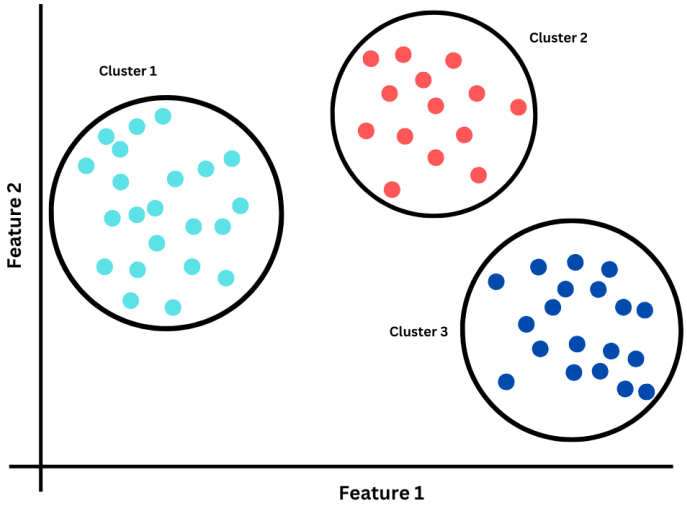
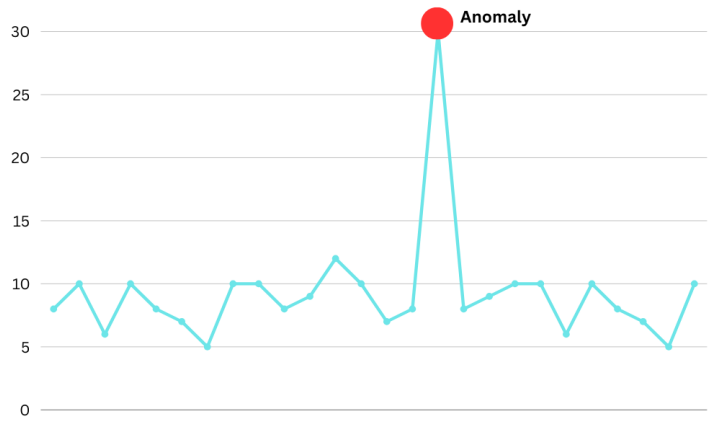


Figure 2.6: Example of Clustering, with three different groups.

- **Anomaly Detection:** Anomaly detection aims to identify data points that deviate significantly from normal patterns. Different algorithms are used to detect unusual sensor readings that may signal impending failures.

Figure 2.7: Example of Anomaly reading of the regular sensory values

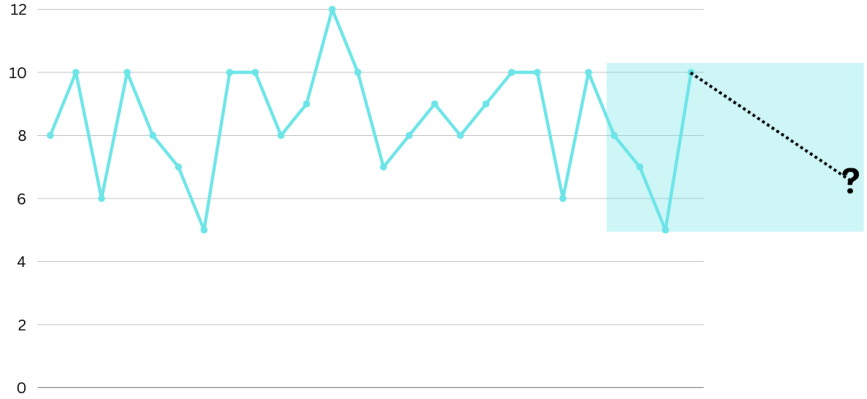


2.6.3 Self-Supervised Learning

Self-supervised learning is an emergent methodology in which models derive insights from unlabeled data by establishing pretext tasks that generate pseudo-labels. This technique is particularly beneficial in scenarios where labeled datasets are limited or incur high acquisition costs.

- **Predictive Task:** Predictive tasks encompass both missing value prediction and next event prediction, enhancing the model's understanding of normal patterns and future outcomes. For example, a model is trained to predict missing sensor readings using partially masked input data. Additionally, the model forecasts future sensor values to identify deviations from expected patterns that may indicate potential failures.

Figure 2.8: Forecasting the future value depending on the previously observed readings (or values)



Chapter 3

Literature review

With the availability of IoT devices and sensor networks, the automotive sector now maintains a log of data [37, 54, 31], either in batch access mode or, in the case of connected cars, through a continuous stream of data. Access to vast amounts of data logs from the vehicles enables companies to conduct real-time monitoring [27, 106] and efficient diagnosis of system components using data analytics and machine learning.

Companies are focusing on using this data to identify patterns through machine learning techniques [49, 33], making the processes of maintenance and repair more efficient. The primary goal in executing these techniques is to apply machine learning algorithms to past data, consisting of vehicle logs and workshop fixes, to correlate and learn specific behaviours that typically lead to machine failure, degraded performance, or a decline in product quality. Predictive maintenance is one of these techniques.

In the context of this work, we incorporate predictive maintenance research, especially in vehicles, into two main areas: (i) Sensor-based outlier detection methods; and (ii) Diagnostic faults-based methods.

3.1 Sensor-based Outlier detection methods

Contemporary systems, particularly within the automotive sector, possess numerous sensors that record various parameters [21]. This data is continuously streamed to disparate systems for analysis, facilitating the discernment of normal behaviour from aberrant behaviour and pattern recognition within sensory readings.

This process is frequently termed Outlier Detection [43] and is a critical component of predictive maintenance, which focuses on identifying abnormal patterns or anomalies in sensor data. A number of research papers cited in the context employ diverse outlier detection methodologies to signal potential faults in industrial systems more broadly [112, 70, 39], with a particular emphasis on vehicles [26, 93]. The sensory data, typically represented in numerical form, is input into anomaly detection algorithms such as clustering [113] and support vector machines [28], which derive meaningful patterns, thereby forecasting the timing of subsequent maintenance requirements.

In addition to the traditional methods mentioned previously, the numerical representation of sensory data allows for the application of a multitude of algorithms aimed at detecting anomalies. For instance, [111] introduces a three-level hybrid model grounded on the median filter, empirical mode decomposition, classification and regression tree (CART), autoregression (AR), and exponential weighted moving average (EWMA) for outlier detection in sensor data.

Ensemble models, which combine weak learners to enhance predictive outcomes, have also been utilized for outlier detection in industrial machinery. [46] illustrates the application of ensemble methods in outlier detection, where a sliding-window-based ensemble method is deployed for streaming outlier detection. This method combines clustering algorithms to form clusters based on data structures, which are subsequently employed in a one-class classification algorithm to ascertain outliers. Similarly, [7] leverage an Ensemble Learning Solution for Predictive Maintenance of

Wind Turbines Main Bearing.

Deep learning techniques have been utilized in the domain of outlier detection. For instance, [71] employed Recurrent Neural Networks to identify anomalies in aircraft data. Another instance, [15], implemented a sparse autoencoder (SAE) network to efficiently identify abnormal data and significantly decrease the false alarm rate. Autoencoders have similarly been applied for anomaly detection in sensory datasets. Furthermore, [105] integrated a deep learning method involving long short-term memory (LSTM) with a one-class support vector machine (SVM) to distinguish abnormal data from normal vibration signals.

Besides parametric models, researchers have adopted non-parametric methods for outlier identification in sensory data; for example, [95] utilized non-parametric techniques such as kernel density estimators to detect distance- or density-based outliers in a single pass over the data, maintaining limited memory requirements. In certain contexts, as noted by [76], non-parametric learning has also been merged with unsupervised learning to perform anomaly detection in machines operating on vibration data.

These approaches demonstrate the application of outlier detection methods in predictive maintenance, highlighting their potential for early fault detection and prevention.

3.2 DTC based Predictive maintenance

With the availability of On-Board Diagnostic systems in vehicles, researchers have shown interest in developing algorithms utilizing diagnostic trouble codes (DTCs), which provide different information about the location and type of the fault. In comparison to sensory data, modelling 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.

Moreover, a significant number of algorithms applied to diagnostic trouble codes (DTCs) rely on repair and warranty data [107] to reformulate the problem into a supervised learning paradigm, which presents challenges in scenarios where such data is absent.

For instance, the predictive maintenance task is often framed as a classification problem [80], employing the Random Forest algorithm to classify data readouts within specified time intervals into faulty and non-faulty categories. By utilizing DTCs as event-based data, a predictive maintenance framework is proposed, delineating four feature categories derived from DTCs. Several studies [100] integrate both onboard diagnostic and historical warranty repair data to forecast warranty repair claims in automotive units. This method involves the comparison of various machine learning algorithms such as Support Vector Machines, Random Forests, and Decision Trees, concluding that the Decision Tree approach demonstrates superior performance. Similarly, the Associative Classification method [22] has been utilized to categorize DTC data.

Examples of approaches focusing on a limited set of DTCs include the application of Random Forest and Long Short-term Memory (LSTM) networks [24] to focus on 6 DTCs associated with the components related to common rail fuel injectors. This study employed one-hot encoding for processing the DTCs due to their limited number, a method impractical for larger sets of DTCs. Another investigation [104] concentrates on a single component, the starter motor, predicting its failure through data mining techniques by developing and assessing three classification models, including the AdaBoost and Random Forest algorithms, based on historical workshop data, error codes, and operational data. Additionally, [75] assesses three algorithms, namely regression tree (CART), C5.0, and C5.0 with boosting, to forecast failures of major automotive systems such as air compressors in long-distance trucks. Neural

networks, particularly long short-term memory networks (LSTMs), have been utilized for DTC-based predictive maintenance, for instance, predicting failures in air compressor components [10] within a specified prediction timeframe. Another example [73] employs signals from the Engine Management System (EMS) CAN-bus (Controller Area Network-bus) as input features with LSTM to classify the pre-DTC sequence period into three pre-selected DTC classes.

Certain researchers have performed feature reduction [89] on DTC data streams using Principal Component Analysis (PCA) before implementing four classification algorithms. They utilize a feedback binary feature representing the overall operational state of the system, with classification tasks executed using four algorithms, including Decision Tree, Random Forest, k-Nearest Neighbors (kNN), and Support Vector Machines (SVM), to determine the operational status of systems such as ignition, exhaust, fuel, and cooling systems. Researchers [20] utilize a binary feature (DTC-status) to apply a Variational Auto-Encoder for computing anomaly scores of DTCs based on their status (active and non-active), enabling engineers to prioritize and focus on the most critical ones.

In terms of understanding and interpretation, Bayesian Belief Networks (BBNs) [45] are employed to guide vehicle diagnostics in a probabilistic framework. In such networks, DTCs are represented as nodes, i.e., cause and action nodes, where the BBN updates node probabilities based on new evidence, generating a novel action list for engineers. Similarly, [47] employs BBNs for vehicle fault isolation. DTC fault event datasets are also utilized to analyze ignition sources in fire cases [50], deviating from traditional fire investigation methodologies. In one study [91], a graphical Bayesian model is integrated with a rule-mining technique for predictive reliability mining to facilitate early warning.

Chapter 4

Sequential Multivariate DTC Fault Event Prediction

The proliferation of onboard diagnostics (OBD) systems in modern vehicles has generated an enormous amount of data on diagnostic trouble codes (DTCs). Predicting the next DTC (next-DTC prediction) is a critical task for vehicle manufacturers, repair shops, and fleet operators to reduce downtime, improve maintenance efficiency, and enhance customer satisfaction. However, this task is challenging due to the complex nature of DTCs, which are often encoded as strings with varying lengths.

As shared in the literature review, studies have demonstrated the potential of machine learning algorithms for DTC fault event based predictive maintenance, but most approaches rely on simplified representations of DTCs, such as one-hot encoding and are constraint to use simple algorithms.

4.1 Overview

In this work, we propose a predictive maintenance approach, named Sequential Multivariate Fault Prediction (SMFP), for predicting the next multivariate DTC fault in an event sequence, using Long Short-Term Memory Networks (LSTMs) and jointly

learned event embeddings. In SMFP, we predict the next DTC event at timestep $T + 1$ given a sequence of multivariate DTC events up to the current timestep T. This enables us to perform predictive maintenance by taking a proactive action of checking or replacing the component predicted by SMFP. In addition, we use multiple features of the predicted event to trace the location and granularity of the fault. We perform an in-depth comparison of different architectural choices and contextual preprocessing techniques, we provide an initial baseline for SMFP that achieves top-3 accuracy of 63% on predicting multivariate fault with 3 collective output layers, using vehicle maintenance data as a case study.

The main contribution and finding of this chapter are:

1. An embedding mechanism, learned jointly with the prediction task, maps multivariate events into a continuous representation space, where similar events are situated close to one another.

2. A multi-output model based on a Long Short-Term Memory (LSTM) network, which is built upon the embeddings, manages the sequential dependencies between the multivariate events while simultaneously optimising the joint event representation space within the embeddings.

3. A baseline for this new methodology is established by conducting an extensive set of experiments with results and observations pertaining to various architectures, embeddings, hyperparameters, and contextual preprocessing approaches for SMFP.

The publication [34] related to this chapter is:

- Hafeez, A.B., Alonso, E. and Ter-Sarkisov, A. (2021). Towards Sequential Multivariate Fault Prediction for Vehicular Predictive Maintenance. 20th IEEE International Conference on Machine Learning and Applications (IEEE ICMLA), pp. 2016-2021, 13- 16 December, Pasadena, CA (virtual).doi: 10.1109/ICMLA52953.2021.00167

In the next section, we define the methodology for learning joint event representations, SMFP with LSTMs, and several architectural choices. In section 4.3, we present results from our experiments. We shall finish with a conclusion in section 4.4.

4.2 Methodology

In this section, we will first share the details about the dataset, the data preprocessing and the problem. We will then discuss how LSTMs handle the sequential dependencies of such multivariate events and how we use them in our work for performing SMFP. Next, we describe how neural embeddings jointly represent multivariate events and how they affect the overall architecture of the model, including the choice between multiple output layers (one for each feature) and a single output layer (with concatenated raw features). Lastly, we compare different contextual preprocessing approaches and their impact on sequential dependency modelling.

4.2.1 Dataset and preprocessing

Diagnostic data, which is located in the memory of electronic modules and control units, is either collected in a diagnostic session or streamed to the cloud periodically. DTC event data for this work is provided by Bosch Automotive Service Solutions [8], which is a leading global supplier of diagnostic, repair shop equipment and spare parts for vehicles. Before preprocessing (grouping) data, a single event row in a dataset corresponds to a fault that occurred in a module of a car. Besides event related information, each row has some attributes related to the car and the session where this event is recorded, for example mileage and time. We initially preprocess the dataset such that all observations (multivariate events) corresponding to one car are grouped into one sequence, ordered by occurrence and mileage. This preprocessing produced 250,000 such sequences. We further preprocessed the data by separating individual features f^i from all the events of a sequence, to form feature vectors $\vec{f}^i$. These two

different preprocessing steps are reflected in figure 4.1 and are used differently for embeddings later.

In figure 4.1-b, we can see a single sequence containing multiple multivariate events from a car. The first feature, which is at the highest granular level, can be seen as the main control source of error like Transmission Module (TM), Object-Detection Module (OD), Break-System Module (BS), etc. The second feature, which is less granular, provides location information such as the part of the vehicle (pedal, chassis area or power area). The third and lowest granular feature can be thought of as fault-type, for example missing the high or low value of some chip. This single arbitrary sequence with the last 5 multivariate events looks like [..., (TM,Power,01), (OD,Camera,12), (BS,Pedal,29), (TM,Power,01), (TM,Battery,101)]. As illustrated in the figure 4.1-b, the goal is to predict the next DTC event with all the three features, given previous multivariate events. Next, we define this process formally.

4.2.2 Sequential Multivariate Fault Prediction

Let $s = (e_T, e_{T-1}, .., e_1)$ be a single such sequence containing multivariate fault events from timestep T to timestep 1 (i.e., in descending order of occurrence), where $e_t = (f_t^1, f_t^2, f_t^3)$ is an event at timestep t with three variables (features) and f_t^i represents the *i-th* feature of this event at timestep t. If we denote a sequence $s = (e_T, e_{T-1}, .., e_1)$ compactly as $s_{T:1}$ and θ be the parameters of the model that we aim to learn, then the candidate event $\widetilde{e}$ from all possible events E that maximises the following probability, is selected as the next predicted event at timestep $T + 1$.

$$\underset{\widetilde{e}}{\operatorname{argmax}}\, P(s_{T+1} = \widetilde{e} | s_{T:1}, \theta) \tag{4.1}$$

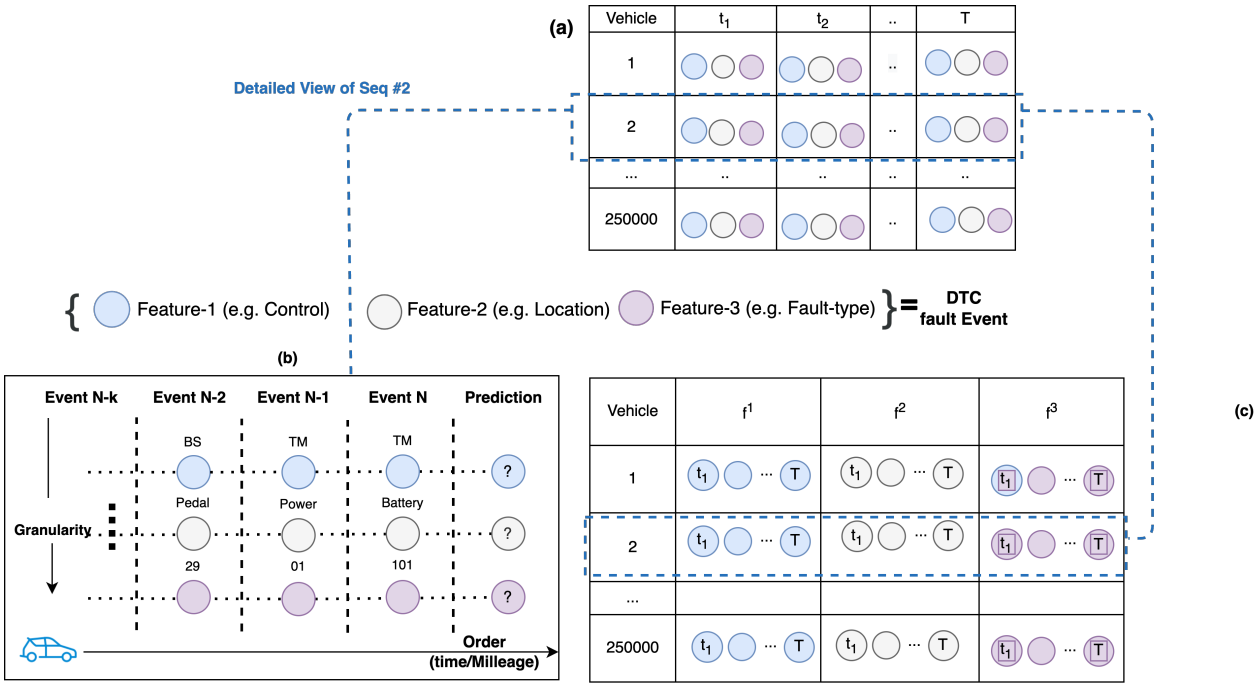


Figure 4.1: (a) An initial preprocessing (grouping) of data resulting in a single sequence per vehicle ordered by occurrence and mileage. (b) A detailed view of a single DTC event sequence having an event containing all three features at each timestep (e.g., the last timestep shows a Transmission Module (TM) related fault in the battery component, with fault type 101). (c) Further preprocessing is done on (a) to separate individual features for applying embeddings.

4.2.3 Sequential dependencies and recurrent neural networks

Since sequences consist of multiple DTC events up to timestep T, it is necessary to capture sequential order and dependencies between events in order to predict the next DTC event. Unlike regular feed-forward networks, Recurrent Neural Networks (RNNs) [19] use a recurrent loop that acts as a memory and helps treat a history of events as a latent hidden state (h). At timestep T, instead of modelling based on a complete sequence $s_{T:1}$, the goal is to efficiently store all the history in the current hidden state h_T and model following

$$P(s_{T+1}|s_{T:1} = (e_T, e_{T-1}.., e_1)) = P(s_{T+1}|h_T) \tag{4.2}$$

The previous hidden state h_{T-1} and the current input timestep s_T contribute to calculating h_T with different learned weights: the weights between the inputs and the hidden layer W_{sh}, the bias weight for the hidden layer b_h and the weights between hidden states W_{hh}. The hidden state h_T is then computed with an activation function a as

$$h_T = a(s_T, h_{T-1}) = a(s_T \cdot W_{sh} + h_{T-1} \cdot W_{hh} + b_h) \tag{4.3}$$

Similarly, we have output weights W_{hq} and a bias weight for the output layer b_q. We use W_{hq} and h_T to calculate the output Y_T as

$$Y_T = a(h_T \cdot W_{hq} + b_q) \tag{4.4}$$

Learning long-term dependencies with RNNs involves multiplying gradients, which results in either very large exploding gradient values or very small vanishing ones [77]. LSTM networks, introduced in [42], are a type of RNNs that addresses the problem of remembering long-term dependencies in sequences by incorporating additional components, namely a forget gate, an input gate, a cell state, and an output gate. Three gates with their respective weight matrices for input layers (W_{sf},W_{si},W_{so}), bias weights (b_f,b_i,b_o) and hidden layers weights (W_{hf},W_{hi}, and W_{ho}) are defined as

$$f_T = a(s_T \cdot W_{sf} + h_{T-1} \cdot W_{hf} + b_f) \tag{4.5}$$

$$I_T = a(s_T \cdot W_{si} + h_{T-1} \cdot W_{hi} + b_i) \tag{4.6}$$

$$O_T = a(s_T \cdot W_{so} + h_{T-1} \cdot W_{ho} + b_o) \tag{4.7}$$

The forget gate (f_T) provides a reset mechanism for the content of a cell state C_T. The input gate (I_T) helps to decide how much information should be read into the candidate cell state $\widetilde{C_T}$, which has a weight W_{sc} with the input, a bias weight b_c and a weight W_{hc} with the hidden layer. The output gate (O_T) channels how much information should the network process, based on C_T. LSTMs prevent unnecessary

updates of content, by avoiding the linear transformation of the cell state C_T and replacing it by point-wise multiplication of the candidate memory and the gates.

$$\widetilde{C_T} = tanh(s_T \cdot W_{sc} + h_{T-1} \cdot W_{hc} + b_c) \tag{4.8}$$

$$C_T = F_T \odot \widetilde{C_T} + I_T \odot \widetilde{C_T} \tag{4.9}$$

4.2.4 Learning to represent events

Representing textual events numerically is required to apply deep learning algorithms and calculate similarity metrics. Some representations are human-interpretable as they use simple metrics like count, presence and absence, etc., of the events. For example, in a Bag-Of-Words (BOW) model [66], the count of occurrences of each word is used as a feature for training. In our case, we can denote a sequence of a feature f^2 with the count of occurrences of its categories (Camera, Pedal, Chassis, etc.). Similarly, One-Hot Encoding (OHE) [82] represents each event in a sequence as an N-dimensional vector, where N is the cardinality of the event. Each vector uses 1 only at the index of that particular word and 0 elsewhere. Unlike BOW, OHE preserves the order of tokens but suffers from the curse of dimensionality (for instance, f^2 of every event at each timestep will be replaced by a 486 dimensional vector, where 486 is the cardinality of f^2). Other types of representation are derived representations, produced by some algorithm or mathematical formula and hence they might not be directly interpretable (for example, Frequency-Inverse Document Frequency (TF-IDF), in [59]).

Neural embedding [87] is a type of derived representation that maps data to continuous low-dimensional representations learned using neural networks. They provide meaningful representations [11] by preserving the similarity of events and mapping each event to a continuous space, such that the events that appear together are close in the new representation. There are many efficient embedding algorithms [88, 5, 96] that produce representations with neural networks, for example, by predicting the

next words given their context (words appearing together) like Word2Vec [99].

4.2.4.1 Representation with neural embeddings

Due to the high cardinality of features, for each feature f^i, a neural embedding matrix ${f^i}_{EMB}$ is learned jointly with a prediction task such that $size({f^i}_{EMB}) < size(f^i)$. A linear projection (${f^i}_{EMB} \cdot f^i_{OHE}$) is applied to obtain a new reduced representation of an OHE *i-th* feature (f^i_{OHE}). In our study, there were two architectural choices for learning joint embeddings (F_{EMB}) for all three features and both have an impact on the number of output layers and on the performance of the resulting architecture. We briefly describe them next.

4.2.4.1.1 Feature concatenation and single entity embedding

In the first approach, we concatenated 3 raw features per time step ($f_{concat} = f^1 \parallel f^2 \parallel f^3$) and used it to calculate an embedding for the event. We also used the OHE of raw concatenation as outputs, which resulted in one output layer, as shown in figure 4.2.

4.2.4.1.2 Multi-input multi-output model with separate embeddings

The drawback of concatenating raw features is that if a feature is missing in the concatenation (f_{concat}), the whole concatenation becomes invalid. This problem reduces the dataset size and the quality of the model and is expected to get worse as new categories arise for each feature. To handle this issue, we changed the approach of preprocessing the data as well as the architecture involving embedding layers. We applied separate embedding layers to individual feature vectors $\vec{f}^i$. After obtaining separate embeddings for each $\vec{f}^i$, all three embeddings were horizontally stacked to form (F_{EMB}), before passing to LSTMs. We then introduced separate output layers for each feature. The architecture for this approach is depicted in figure 4.3.

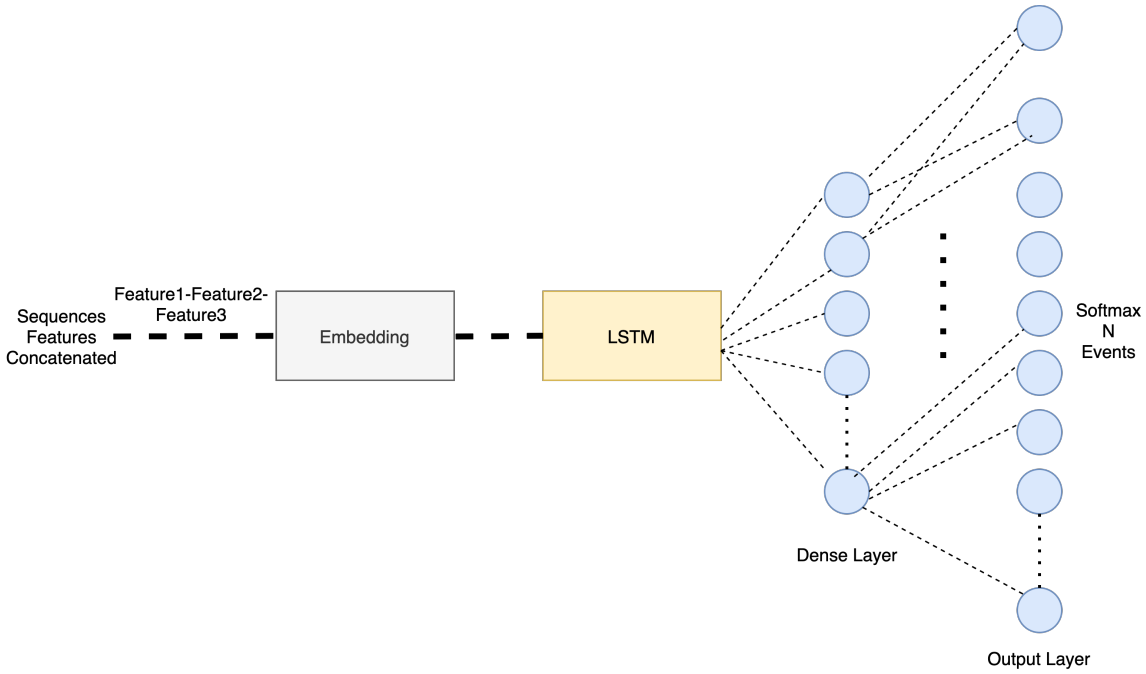


Figure 4.2: Architecture diagram with a single embedding for concatenated features and a single output.

Since weights for these embeddings are learned jointly with the downstream prediction task using LSTMs, they are optimized such that they retain sequential and semantic similarities in the learned continuous representation space.

4.2.5 On context length and its influence

Theoretically, LSTMs can handle sequences of variable lengths with long-term dependencies without encountering vanishing or exploding gradients. Despite this, due to the large variations in the length of sequences and computational constraints, the number of events to consider (i.e., context) to predict the next event is an important decision for the architectural choice of an LSTM network. Moreover, within a single sequence corresponding to one vehicle, there can be different unrelated event sub-sequences. For example, if we aim to predict the next DTC event in a sequence containing 15 total events, it is possible that only the last 7 events (a sub-sequence

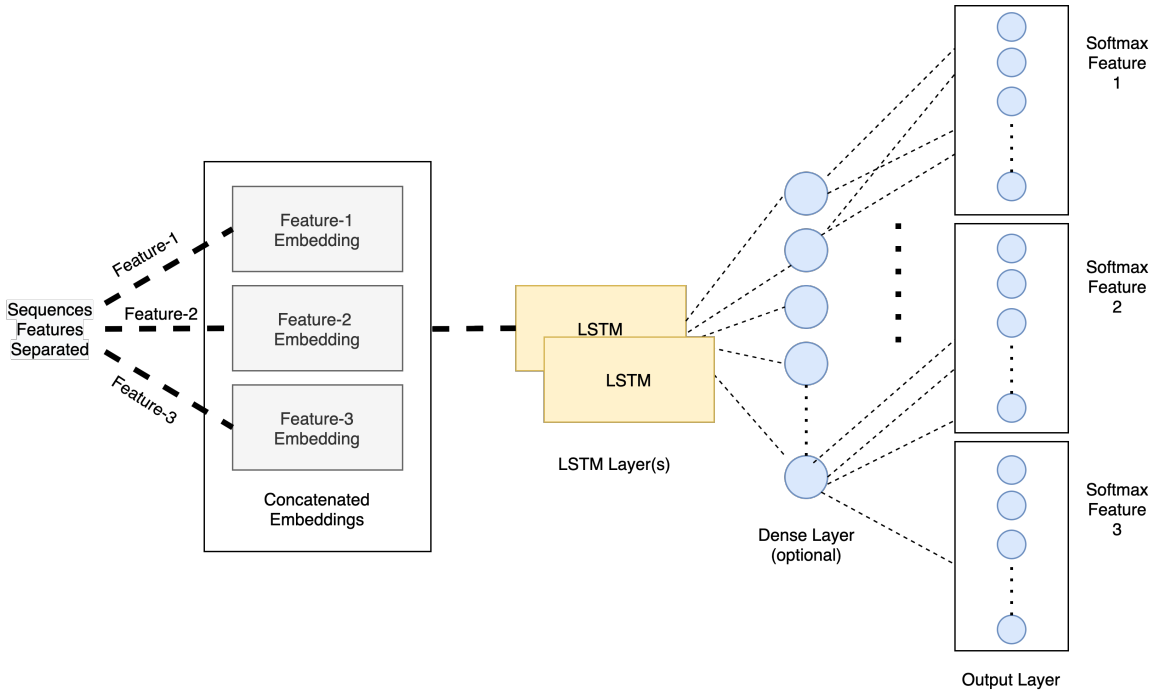


Figure 4.3: Architecture diagram for the approach of concatenating separate embeddings for every feature and multiple-outputs.

from t_8 to T) are related and will result in the failure of power. We compare taking only the last N events to different sequence preprocessing techniques, in order to ascertain if an LSTM can learn these relations without additional context.

Using the *Fixed Window Split* technique [79], we divided long sequences into separate sequences after a fixed window size of N timesteps. For instance, if one sequence has a length of 80, it will be divided into 8 sub-sequences with a split size of N=10. The *Sliding window* approach is used in many areas of information theory such as in time-series prediction [25] and NLP. In our study, for each sequence, we took a fixed window of N timesteps of a sequence at a time and then slid this window to the right with stride size 1.

We also used *Overlapping windows*, where for each window of size N, instead of sliding it with stride size 1, we included the last M timesteps of the previous window into the window, giving an effect of smooth overlap and transition. Finally, we used

Last-N Timesteps, where we considered only the last N timesteps of each sequence. In most of the approaches mentioned above, we had some sequences which were shorter than the window size and had to be padded by 0.

4.3 Experiments and Results

Our model predicts a multivariate event using three dense output layers. We use OHE to represent actual output features (ground truth labels) of an event and a softmax activation function [9] in output layers. These layers provide K_{f^i} probabilities, where K_{f^i} is the cardinality of a particular feature f^i. All output layers have cross-entropy loss [64] as the measure of errors on predicted outputs. The overall loss is the sum of the three individual losses for each feature f^i. If $\hat{y}$ denotes the predicted class, y corresponds to an actual class, and K_{f^i} is the size of the output layer for a particular feature, the overall categorical cross-entropy loss [29] of all F features is calculated as

$$L(\hat{y}, y) = \sum_{f^i}^{F}(-\sum_{k}^{K_{f^i}} y^{(k)} \log(\hat{y^{(k)}})) \tag{4.10}$$

Binary classification [53] and multi-class classification [30] problems use *accuracy* to evaluate the performance of the model. This metric measures the fraction of correct predictions out of total predictions. Due to the high cardinality of features (70, 486, and 84, respectively), we used a top-3 accuracy metric. This metric measures if the actual values of all three features fall in their respective top-3 predictions, which are probabilities sorted in descending order. For example, imagine that the actual fault that occurred was Camera (f^2 in the event) and that the model assigns the highest probabilities to Battery, Chassis, and Camera in descending order (i.e., Camera is in the top-3 probabilities for this feature f^2): if the same stands true for the other two features, then we count it as a true prediction.

4.3.1 Experimental setup and hyperparameter tuning

We have trained our model using 250,000 sequences, each of which corresponds to a unique vehicle. We performed training and test split of the dataset before applying any contextual preprocessing technique. We kept aside 12,500 sequences for testing, and 4,750 sequences for validation, and used the remaining 232,750 sequences for training. Adam optimizer, with a learning rate of 0.001, was used for optimizing the network. Data preprocessing, including ordering the sequences with mileage and time, is done using Apache Spark [3] while the rest of the implementation uses Keras [13].

Hyperparameter choices, which we explain in detail in individual result sections, include (i) using one wide LSTM layer instead of two LSTM layers, (ii) using 60 to 240 units in each LSTM layer, (iii) whether to use a dense layer on top of the LSTM layer (iv) dropout values (both recurrent dropout and dropout after LSTM layers), and (v) the output size of the embedding (5 to 20 percent of the original feature cardinality) for each feature. All hyperparameters are selected using Keras Tuner, which provides different hyperparameter tuning algorithms like Random Search and Bayesian Optimization [92]. Common hyperparameter choices are listed in Table 4.1.

Table 4.1: Common hyperparameter choices

Choice	Values
Optimizer	Adam (beta-1 = 0.9 & beta-2 = 0.999)
Learning rate	0.001
Dropout & Recurrent Dropout	0.1 to 0.3

4.3.2 Comparing feature concatenation with separate embeddings and selecting embedding dimension

Before running all experiments with the different hyperparameters and architectural choices, we evaluated the performance of two basic feature representation mechanisms; concatenating all three features before applying embedding and concatenating separately learned embeddings of each feature. The first row in Table 4.3.2 shows the performance of SMFP with a single concatenated input feature and a single output, while the second row reflects the result of a multi-input and multi-output model. We achieve 63% top-3 test accuracy for the separate embedding concatenation approach, while the alternative approach resulted in 51% top-3 test accuracy. Besides the difference in the results, the embedding concatenation approach is also flexible for feature preparation, as explained in 4.2.4, so we have used separate feature embeddings in all other experiments.

The impact and choice of embedding dimensionality, which is currently a much-focused research area [109], depends on the cardinality of the feature being embedded. Although embeddings avert the curse of dimensionality, a very low dimension can also fail to find the latent features. With hyperparameter tuning, 10% of the original feature cardinality (e.g., the cardinality of $f^2 = 486$ maps to $486 \cdot 0.10 \approx 48$ dimensions vector) was selected as the embedding dimension for the experiments.

Representation Type	Top-3 Accuracy
Concatenated Features and Embedding	51%
Separate Embeddings	63%

Table 4.2: Results of single embedding of concatenated features and separate feature embeddings, while keeping the other configurations constant

4.3.3 Contextual preprocessing approaches

We experimented with the multiple contextual preprocessing designs, discussed in section 4.2.5. We preprocessed all the 250,000 sequences. Each preprocessing approach resulted in a different number of sub-sequences. For a single sequence corresponding to one vehicle, while the Last-N Timesteps approach produces only one sub-sequence, the Sliding Window and Overlapping Window approach produce a large number of sub-sequences due to overlapping repeated events and hence they start to overfit very early. We compared these alternatives to ascertain which one captures the context better. As shown by Table 4.3.3, the Last-N Timesteps method reached the lowest validation loss of 4.621 and the highest top-3 test accuracy of 61%.

Having good performance using Last-N Timesteps suggests that the LSTM can learn dependencies between events as well as sub-sequence breakpoints within sequences, which is a concern that we discussed in section 4.2.5.

4.3.4 Using dense layer on top of LSTM layers

We also added a dense hidden layer (with 120 neurons) after the LSTM layers to the best-performing preprocessing (Last-N Timesteps) to see if it would help in learning more relations. But as shown in Table 4.3.3, it didn't result in an improvement in accuracy or validation loss. Despite the increment in the number of parameters, top-3 test accuracy scored 62% and validation loss 4.60.

4.3.5 Dropout and recurrent dropout

Dropout, originally introduced in [41], is a regularization technique used in deep learning to prevent overfitting and improve generalization. For each training iteration, the dropout randomly selects a set of neurons to be deactivated. It prevents the network from relying only on a few units and forces all units to learn independently. In [23], it is argued that applying the same dropout mask at each time step is more efficient

Method	LSTM Layers	LSTM Units	Dense Layer	Dense Units	Embedding Output Dimension	Drop out	Train Loss	Valid Loss	Top-3 Test Acc
Last-N Timesteps	1	120	0	-	$10\% \cdot size(f^i)$	-	4.44	4.62	0.61
Last-N Timesteps	2	64+84	0	-	$10\% \cdot size(f^i)$	-	4.42	4.63	0.60
Last-N (with Dense Layer)	1	120	1	120	$10\% \cdot size(f^i)$	-	4.38	4.60	0.62
Last-N Timesteps	1	120	0	-	$10\% \cdot size(f^i)$	0.2,0.2	4.58	4.52	0.63
Fixed Window	1	120	0	-	$10\% \cdot size(f^i)$	-	5.43	5.20	0.50
Sliding Window	1	120	0	-	$10\% \cdot size(f^i)$	-	5.37	5.33	0.47
Overlapping Window	1	120	0	-	$10\% \cdot size(f^i)$	-	5.38	5.57	0.49
Overlapping Window	2	64+84	0	-	$10\% \cdot size(f^i)$	-	5.39	5.61	0.48

Table 4.3: Results with multiple contextual preprocessing approaches, different hyperparameters and architectural choices

than applying it randomly. The authors also suggest that recurrent activations should be masked with a constant mask, which is often called recurrent dropout.

In Table 4.3.3, it can be seen that, without dropout, the validation error stopped decreasing in all contextual preprocessing variations, while the training loss was still lower than the validation loss, which showed overfitting in the network. Using recurrent dropout and dropout of 0.2, which means deactivating 20% of the units randomly in the LSTM layer at the time of training, helped controlling overfitting in the network. These dropouts produced the lowest validation error of 4.52 among all the experiments.

4.3.6 Comparing LSTM units and wider-deeper networks

The number of LSTM units corresponds to the size of the hidden state, where one state captures one latent feature, for example, the presence or absence of an event that leads to engine malfunction. For the first layer of the LSTM, a final value of 120 units in a single LSTM layer was selected using hyperparameter tuning. One of the architectural choices for this study was to use a wider network instead of a deeper network [2] by adding more units in the first layer instead of stacking a second LSTM layer on top of the first layer. We noticed that the wider network showed the same or slightly better performance than the deeper network. These results were consistent across all preprocessing approaches in Table 4.3.3. To perform a balanced comparison for two-layered LSTMs, we kept the number of LSTM parameters close to the single-layered experiment, using 64 units in the first LSTM layer and 84 neurons in the second layer.

4.4 Discussion and Conclusion

In this chapter, we propose a new event-based predictive maintenance approach, Sequential Multivariate Fault Prediction (SMFP), which differs from the anomaly detection approaches that rely on sensor data having numeric representations. We show how complex multivariate events, which are non-numeric, can be mapped to continuous representations by jointly learned embeddings. We propose an LSTM-based architecture that uses these representations for SMFP.

Furthermore, set a baseline for SMFP and showed various contextual preprocessing approaches and architectural choices including multiple-output modelling, embeddings on raw feature concatenations, and stacking separate embedding layers. Our experiments achieve a baseline of 63% top-3 test accuracy for SMFP. We aim to further improve these results by trying other algorithms such as Seq2Seq models and Attention methods [97],[102].

Chapter 5

Using learned DTC representation for Interpretation, Search and Exploration.

In our previous chapters, we proposed the SMFP model, which utilised LSTMs to predict the next DTC event in a sequence of multivariate events [34]. We demonstrated that the SMFP model achieved state-of-the-art results in terms of accuracy and efficiency. However, there were some limitations associated with this approach. In particular, the SMFP model did not offer any insight into how the predictions were made or which individual DTCs and sequences contributed to its overall performance. Moreover, retrieving specific DTCs and sequences from the predicted output presented a challenging task.

To address these limitations, we propose an extension of the SMFP model, called DTCEncoder. The DTCEncoder architecture uses neural embeddings and a recurrent model, taking inspiration from the SMFP model while incorporating additional components that facilitate the interpretation of results and efficient retrieval of individual DTCs sequences. In this chapter, we will introduce the DTCEncoder architecture in detail and demonstrate its effectiveness in enhancing accuracy and efficiency.

5.1 Introduction

This chapter presents an attention-based DTCEncoder, which aims to learn low compact representations of multivariate event sequences and hence enable interpretable multivariate DTC event prediction. The contributions of this chapter are:

- A novel attention-mechanism based DTC sequence encoder, which produces a compact representation of DTC sequence. These representations explicitly incorporate information from all previous hidden states according to their importance
- Improved performance of the next-DTC prediction method, along with increasing interpretability.
- Demonstration on how compact representation can be used to reduce search space for the semantic search of multivariate sequences, and performs event and sequence level retrieval.

Following publication [36] is related to this chapter:

- A. B. Hafeez, E. Alonso and A. Riaz, "DTCEncoder: A Swiss Army Knife Architecture for DTC Exploration, Prediction, Search and Model Interpretation," 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA), Nassau, Bahamas, 2022, pp. 519-524, doi: 10.1109/ICMLA55696.2022.00085.

In section 5.2, we present the methodology 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.

5.2 Methodology

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

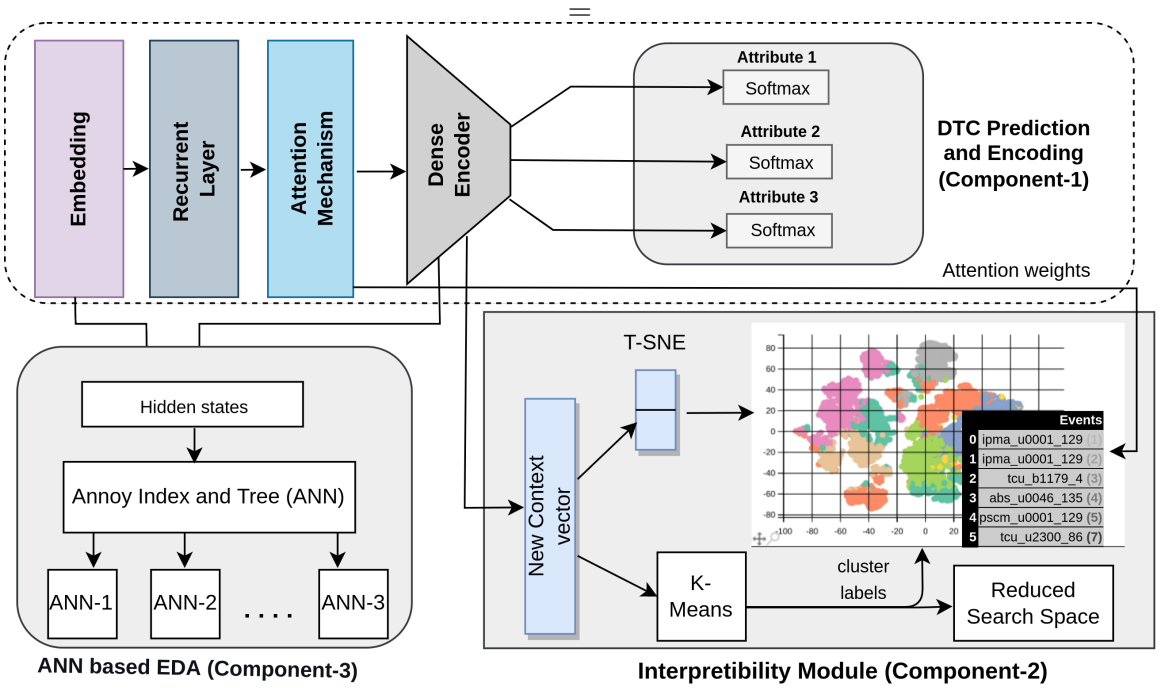


Figure 5.1: The overall architecture of the DTCEncoder. Component-1 on top predicts the next DTC with the encoding unit, which learns a low dimensional representation for each DTC sequence with the help of the attention mechanism, the GRU, and the dense encoder. The interpretability module on the bottom right (Component-2) utilizes context-vectors and attention-weights produced by component-1, to perform dimensionality reduction and visualization. Component-3 on the bottom left 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.

5.2.1 DTCEncoder motivation

In SMFP [34], we 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 [63] 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 [72] 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 of DTC sequences (and individual DTCs). The overall architecture of the DTCEncoder can be seen in figure 5.1 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**
    $a_{EMB}^1, a_{EMB}^2, a_{EMB}^3 \leftarrow EMB(a_{OHE}^1, a_{OHE}^2, a_{OHE}^3)$
    $a_{EMB} \leftarrow a_{EMB}^1 \parallel a_{EMB}^2 \parallel a_{EMB}^3$ {Concatenating the individual embeddings}
    $gru\_state \leftarrow GRU_2(GRU_1(a_{EMB}))$
    $lastSt$, $stWithoutLast \leftarrow slice(gru\_state)$
    $Score$,$Context \leftarrow Attention(lastSt, stWithoutLast)$
    $Dense1 \leftarrow Dense(Context)$
    $DenseBottleneck \leftarrow Dense(Dense1)$
    $a_{pred}^1 \leftarrow Dense(DenseBottleneck)$
    $a_{pred}^2 \leftarrow Dense(DenseBottleneck)$
    $a_{pred}^3 \leftarrow Dense(DenseBottleneck)$
    Calculate loss
    Optimize parameters of all layers
  end for

5.2.2 Attention mechanism and Encoder

Unlike SMFP where prediction for the next DTC event is made using the last hidden state of the LSTM, we use the Loung attention mechanism [61] to obtain a more representative context-vector by taking a weighted sum of the hidden states from all time steps, where the weights are calculated with an attention mechanism.

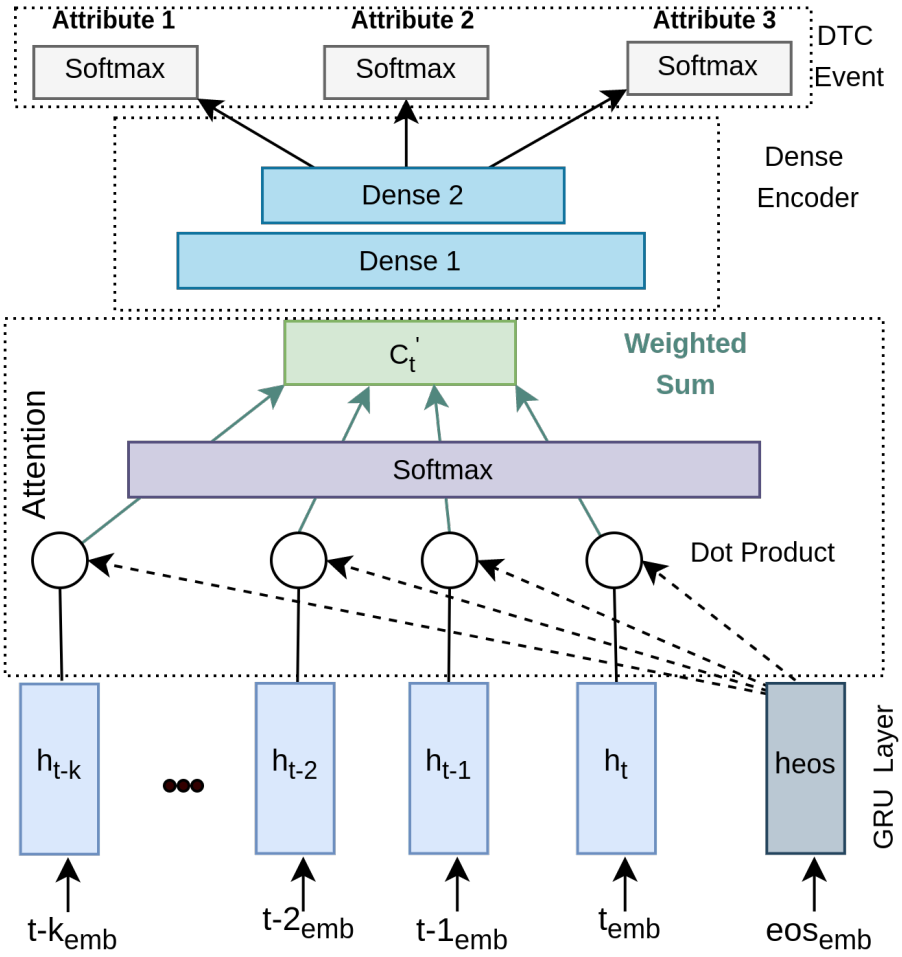


Figure 5.2: The attention mechanism performs a dot product operation between all hidden-states h_t and the EOS hidden-state. The new context-vector $\hat{c}_t$ is obtained by performing the weighted sum of all hidden states. The new context-vector is passed through dense layers with a decreasing number of neurons (Encoder), and finally to the output layers of all three attributes.

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 5.2 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

$$\alpha_{t,j} = \frac{exp(e_{t,j})}{\sum_{i=1}^{T} exp(e_{t,i})}, \tag{5.1}$$

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

$$\hat{c}_t = \sum_{i=1}^{T} \alpha_{(t,i)} \cdot h_i \tag{5.2}$$

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 the final hidden dense layer (bottleneck) is then passed to the three dense output layers, each corresponding to a different attribute of the DTC event.

5.2.3 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 [62] to this 24-dimensional representation to further reduce it to 2 dimensions, which can thus be visualized.

As seen in component-2 of figure 5.1, the interactive module of the DTCEncoder applies the K-means clustering algorithm [38] 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 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 5.3, 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 5.3, 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.

Algorithm 2 ANN with Annoy [94] 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 $\leftarrow$ build Index for all Sequences (or DTC events)
3: Return Index

Algorithm 3 Clustering and t-SNE

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

5.2.4 Exploratory analysis and semantic search of DTCs and DTC sequences

It is often 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) [58] 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 5.1 provides a visual flow of the Indexing and exploratory analysis pipeline.

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

Table 5.1: Top 3 neighbors of frequent occurring DTC faults

We build an ANN Index with an ANN algorithm called *Annoy* [94]. Annoy uses

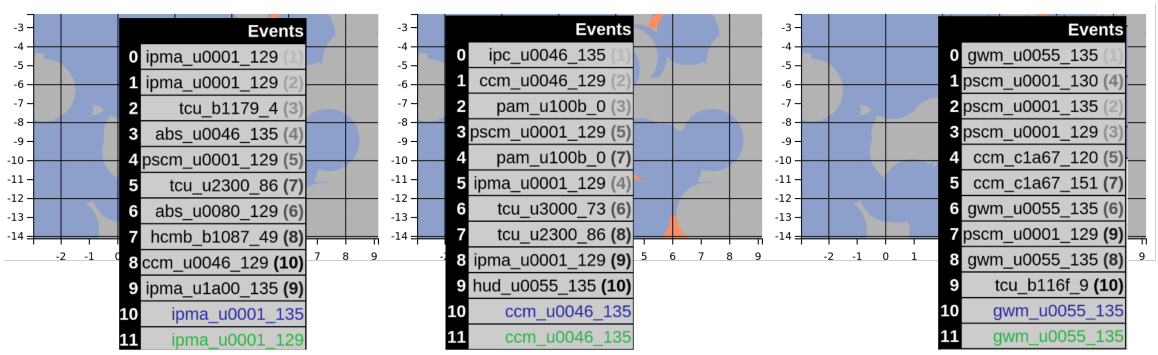


Figure 5.3: The interpretability module maps sequences to a 2-dimensional t-SNE space and color codes them according to the K-means clusters learned on their context-vector. This figure shows three different sequences in proximity by hovering different positions in the interpretability module. Each sequence 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.

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. 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 5.2.4 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).

For sequences, we learn an Annoy Index on the 24-dimensional bottleneck repre-

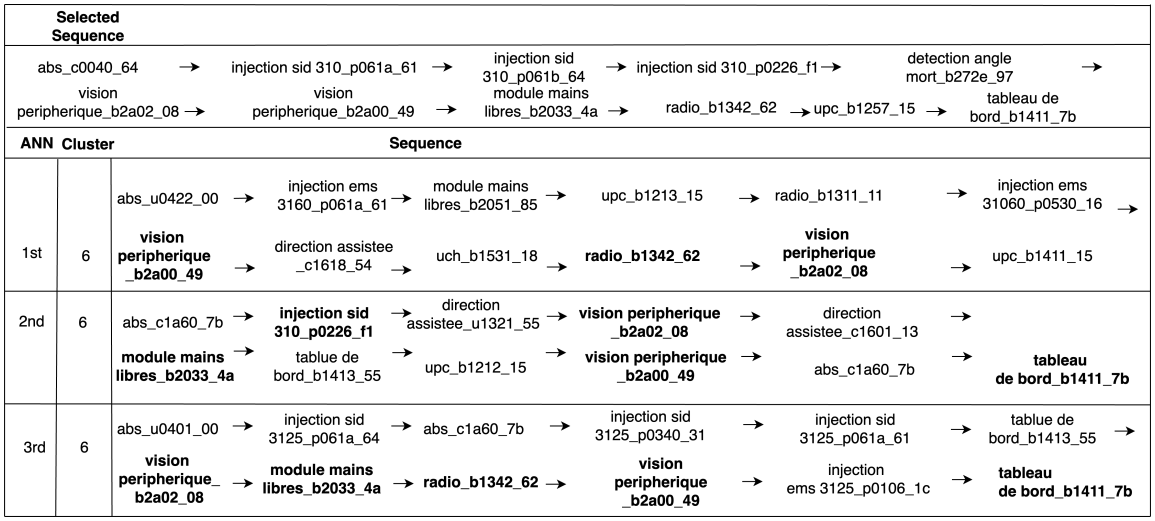


Figure 5.4: Example of three similar sequences provided by ANN, for a selected DTC. DTCs which are common in the selected DTC and similar DTCs are highlighted in bold. For example, the selected sequence, the second, and the third similar sequence all contain specific DTCs (e.g., vision peripherique_b2a02_08) and end in the same DTC event (tableau de bord_b1411_7b)

sentation (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 5.4, the most similar sequences retrieved by the Index are the ones that have more DTCs in common or end in the same faults (DTCs).

5.3 Experiments and Results

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

5.3.1 Dataset

We used the same dataset provided by Bosch Automotive from [3] to train the DTCEncoder. As defined in section 5.1 and depicted in figure 5.5, each sequence contains multivariate fault events and corresponds to unique vehicles. Out of 250,000 fault sequences, we used 232,750 sequences for training, 12,500 separate sequences for testing, and 4,750 different sequences for validation.

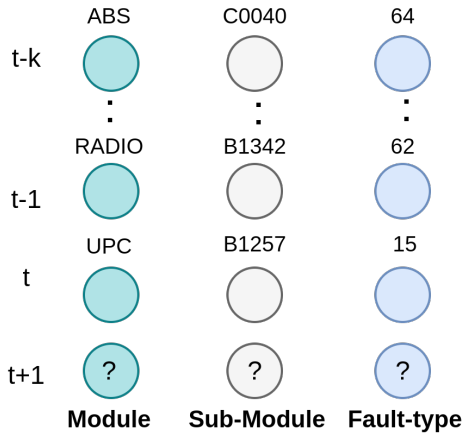


Figure 5.5: Single sequence showing multivariate DTC events from time t-k to time t. The goal is to predict the multivariate event at time $t+1$.

5.3.2 Experimental setup and hyperparameter tuning

The hyperparameters and model parameters defined in this section are opted with the Hyperband [55] hyperparameter tuning method using keras-tuner [74]. The attribute embedding sizes that we considered, along with the sizes selected by tuner, are mentioned in table 5.2. 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 5.2. We used Adam optimizer [51] with a learning rate of 0.0066.

Table 5.2: 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

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.

5.3.3 Results

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%

Table 5.3: Ablation study and results of DTCEncoder in comparison with the SMFP's.

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 introduced previously, since, as far as we know as of writing this chapter, there is no other model that performs such a task.

In earlier chapters, Top-3 accuracy was used as an evaluation metric to report the model's predictive performance. From this chapter onward, Top-5 accuracy is adopted to provide a more appropriate assessment, particularly in light of the large number of output classes. This change does not alter the evaluation methodology but reflects an effort to offer a more comprehensive view of model performance. The adjustment was made in response to academic feedback highlighting that Top-3 accuracy alone may underrepresent the model's predictive effectiveness in multi-class prediction scenario, where the number of unique classes are quite high.

We achieved the lowest validation loss of 4.36 and top-5 accuracy of 79% with our architecture, in contrast to a 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 the output layers. As shown in table 5.3.3, 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.

5.4 Discussion and Conclusion

In this chapter, we have presented a unified architecture 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 shared in the previous chapter.

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. This model makes the next DTC prediction mechanism interpretable with a unified architecture and automatically improves the performance of the other components (e.g., interpretation, semantic search) 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.

Chapter 6

Hybrid model to improve the next-DTC prediction accuracy with Transformer and GRU

In preceding chapters, we presented architectures incorporating embedding components and sequential networks capable of managing the sequential and multivariate dependencies present in DTCs. The SMFP model employed LSTMs to address these complexities, while the DTCEncoder utilized the attention mechanism to further enhance accuracy and provide a dense, reusable representation of the complete DTC sequence.

Nevertheless, as previously noted, this methodology is constrained by limited datasets. Our proposed DTC-TransGRU architecture seeks to mitigate these challenges by integrating a small transformer model with a GRU network. This innovative approach enables the model to effectively capture both short-term and long-term dependencies, while also achieving greater computational efficiency compared to large, standalone transformer models.

6.1 Overview

There has been an increase in the adoption of recently introduced sequential models in other domains, like transformers, because of their strengths. These model require a lot of data and computation power for training, which is not in abundance in all cases. In the DTC-based research, we came across the use of a Transformer-decoder [81] approach with a small-GPT-2 [84] architecture along with embeddings to perform the DTC prediction task. Running our data through such a large model resulted in very bad top-5 accuracy, which resonates with the results of the paper. This motivated us to look at exploring a smaller but hybrid model, which can benefit from both the representation capability of the transformer, along with used a different model to compensate for the weakness of the small transformer model.

This chapter presents a hybrid model, where we combine Transformer [103] and GRU [12] 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, in multiple ways with the help of multiple heads. Moreover, GRU reinforces the model's understanding of the temporal dynamics in the sequence.

Following publication [35] is related to this chapter:

- Abdul Basit Hafeez, Eduardo Alonso, and Atif Riaz. 2024. DTC-TranGru: Improving the performance of the next-DTC Prediction Model with Transformer and GRU. In Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24). Association for Computing Machinery, New York, NY,

USA, 927–934. https://doi.org/10.1145/3605098.3635962

The rest of the chapter is structured as follows: in section 6.2 provides details about the methodology used by DTC-TranGru for the next DTC prediction task. The experiments and results are presented in section 6.3. We close the paper with a discussion of our approach and future work.

6.2 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.

6.2.1 DTC-TranGru Model

The overall architecture of the DTC-TranGru model is shown in figure 6.1 and the pseudocode is provided in algorithm 4. It is composed of the following main components.

6.2.1.1 Embedding Layer

Due to the high cardinality of DTC event's attributes, we introduced and used neural embeddings to learn low-dimensional representations of attributes in the previous chapters. In our model, we start similarly 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 [110], with a dropout rate of 0.1, to these combined embeddings.

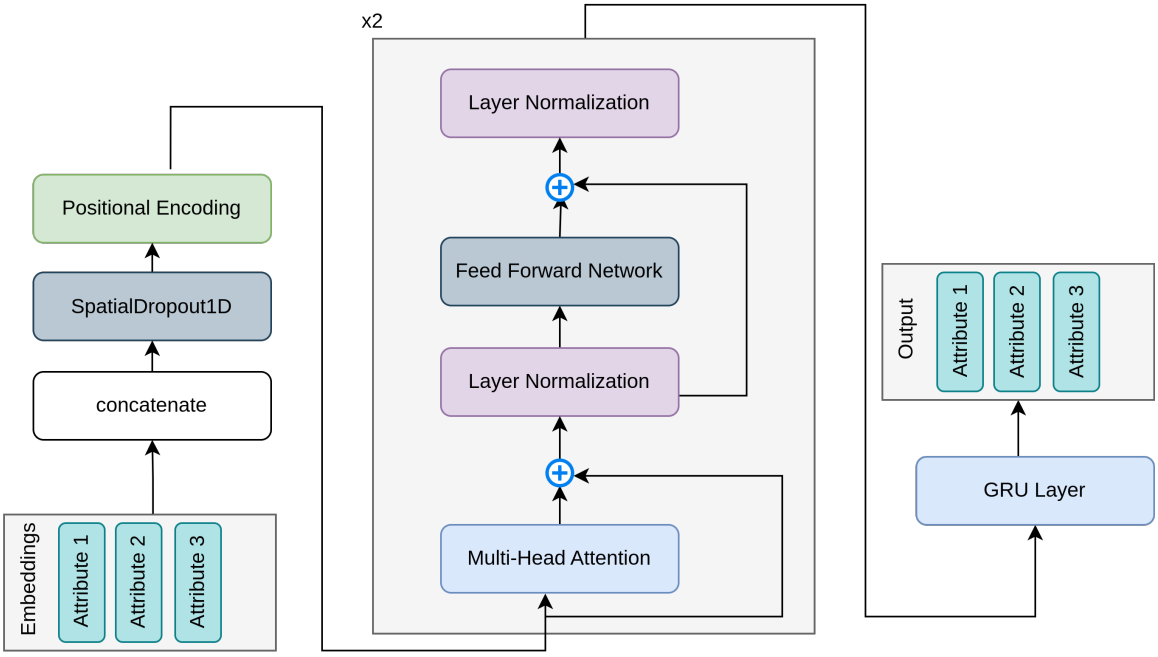


Figure 6.1: Architecture diagram for the DTC-TransGru. The left side of the figure shows pre-transformer operations, where DTC-TransGru starts by applying separate embedding layers to each attribute, and concatenates the individual embedding layers before passing them to spatial-1d dropout [110]. In 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.

6.2.1.2 Positional Encoding

The transformer layer, which will be detailed in 6.2.1.3, 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 [18], to be used with the transformer layers, to keep 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 [103], which utilizes sine (sin) and cosine (cos) functions to create po-

sition 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.

6.2.1.3 Transformer Layer

As opposed to SMFP, 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:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{6.1}$$

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

$$\text{head}_i = \text{Attention}(QW_{Qi}, KW_{Ki}, VW_{Vi}) \tag{6.2}$$

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

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_n)W_O \tag{6.3}$$

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 [48] by ensuring that gradients can flow smoothly, preventing issues like vanishing gradients, and allowing deep networks to learn effectively. Similarly, a layer-normalization [108] 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 6.2, 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:

$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2 \tag{6.4}$$

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.

6.2.1.4 GRU Layer

As shown in figure 6.3, 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 6.3, 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

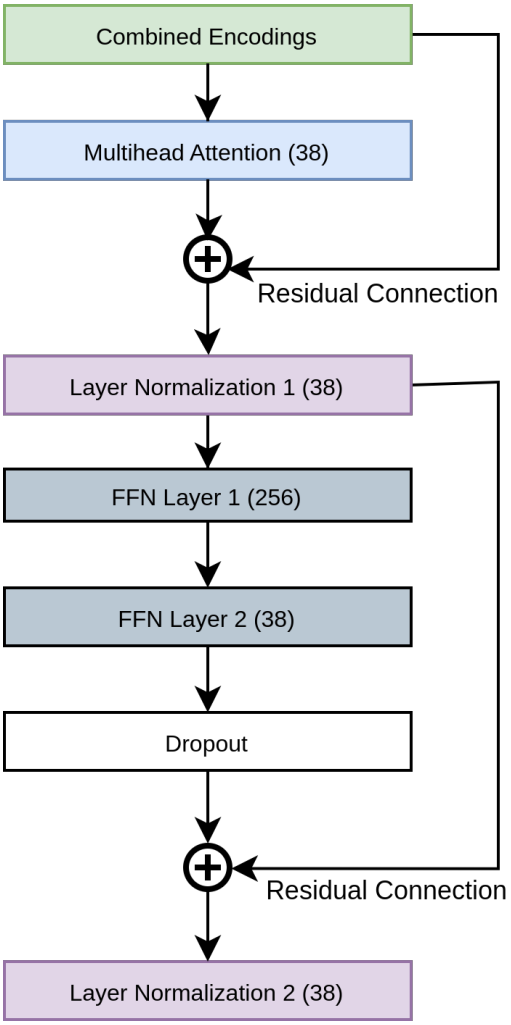


Figure 6.2: Detailed view of the encoder layer of the DTC-TranGRU's transformer block. Each layer shows its output dimension next to its name. It can be seen that there are two residual connections, two dense layers in FFN, and two layer-normalization layers. The dimension of the output is scaled up 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.

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.

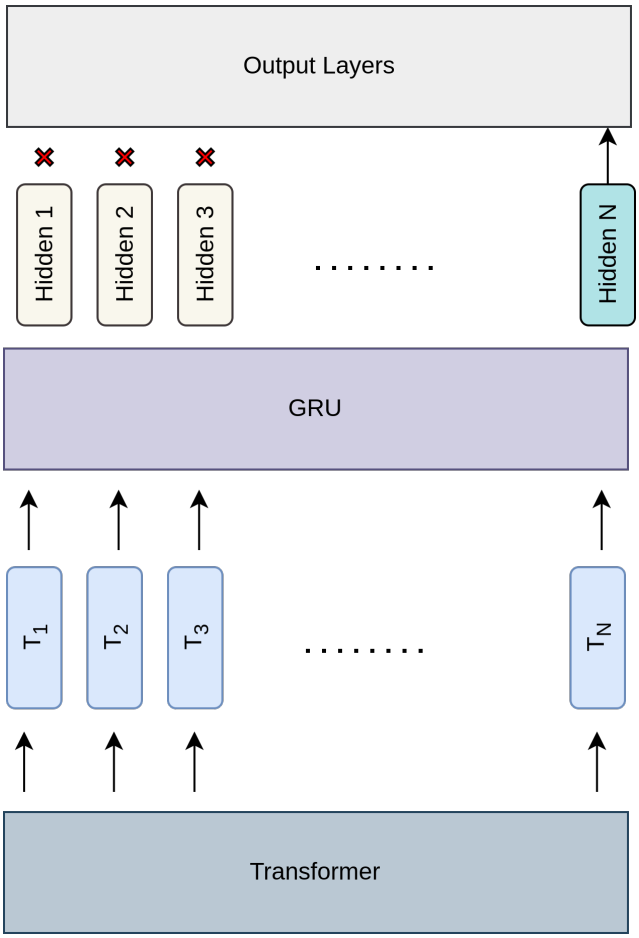


Figure 6.3: The transformer layer returns N timestep outputs (each containing *EMB-SIZE* latent dimensions), which typically are averaged, summed, or go through the GlobalAveragePooling1D layer to make it compatible with the dense output layer. Instead of doing this, we applied a GRU layer on top of 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.

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

Algorithm 4 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**
      $multihead \leftarrow multihead_n(enc)$
      $l\_norm\_1 \leftarrow Layer\_Norm(pos\_enc + multihead)$
      $ffn\_l1 \leftarrow Dense(multihead)$
      $ffn\_l2 \leftarrow Dense(ffn\_l1)$
      $ffn\_l2 \leftarrow Dropout(ffn\_l2)$
      $l\_norm\_2 \leftarrow Layer\_Norm(ffn_l2 + l\_norm\_1)$
      $enc \leftarrow l\_norm\_2$
    end for
    $GRU\_OUTPUT \leftarrow GRU(enc)$
    $attr^1_{pred} \leftarrow Dense(GRU\_OUTPUT)$
    $attr^2_{pred} \leftarrow Dense(GRU\_OUTPUT)$
    $attr^3_{pred} \leftarrow Dense(GRU\_OUTPUT)$
    $loss \leftarrow$ ComputeLoss($attr^1_{pred}, attr^2_{pred}, attr^3_{pred}$)
    OptimizeParameters($loss$)
  end for

transform to the output layers but also learns contextual dependencies associated with the events.

To calculate the loss and performance of the model, we used cross-categorical loss [29] for the output layer of each attribute and summed the individual loss of all three attributes with equal weights as follows

$$L(\hat{y}, y) = \sum_{a^i}^{A}(-\sum_{k}^{K_{a^i}} y^{(k)} \log(\hat{y^{(k)}})), \tag{6.5}$$

where K_{a^i} is the number of unique classes in a given attribute a_i, A represents the

number of total attributes, $\hat{y}$ represents the predicted class, and y denotes the actual class.

6.3 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.

6.3.1 Dataset and Data Preprocessing

This research used vehicular DTC sequence data provided by *Bosch Automotive*. 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 6.4, 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.

6.3.2 Experimental setup and hyperparameter tuning

The hyperparameters and parameter choices, which are shown in table 6.3.2, are selected by running the Hyperband [55] hyperparameter tuning method available in the python keras-tuner [74] library.

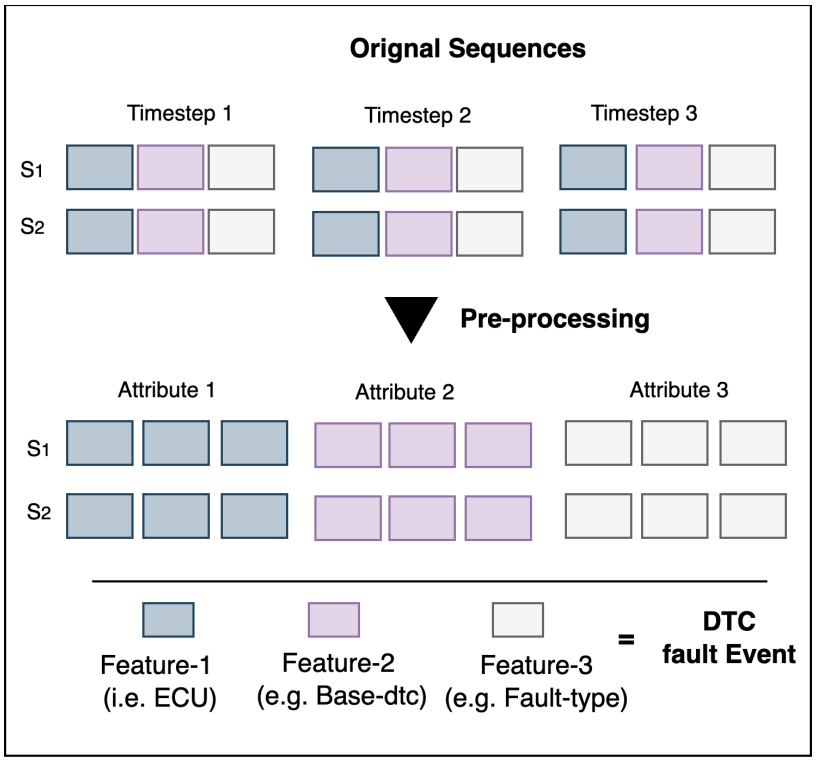


Figure 6.4: An example of two DTC fault sequences with 3 DTCs each, undergoing the preprocessing step. Each attribute is vectorized and separated so that it can then be passed to its independent embedding layer. The choice of the number of events is just for the sake of illustration, otherwise, all of the DTC sequences used in this experiment consist of 5 DTC events at least.

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 [51].

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.

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-3 (fault-byte) embedding	4	32	8
Spatial dropout after embeddings	0.0	0.5	0.1
GRU layer units	96	256	128

Table 6.1: 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.

We used RELU [69] 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.

6.3.3 Results

Table 6.3.3 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 introduced in the previous chapter, 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 introduced in chapter 4, as a part of the ablation study.

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%**

Table 6.2: Comparison of the results achieved by DTC-TranGru compared with SMFP, DTCEncoder, [81] 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.

Table 6.3.3 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 6.3.3, 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 [81], which is based on a smaller version of the GPT-2 model [84] (often referred to as small-GPT-2), would overfit. But for the sake of completeness and accurate comparison, we modified the model used in the Transformer-Decoder [81] 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 6.3.3, 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.

6.4 Discussion and Conclusion

In this chapter, we present a transformer and GRU-based architecture to improve the performance of the next DTC prediction task. In the proposed architecture, 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.

Chapter 7

Enhancing predictability with Optimized connections in Transformer-GRU architecture and with Ensemble of models.

In the preceding chapter, we introduced a model based on the Transformer-GRU architecture, which incorporates a GRU layer subsequent to a compact transformer network, with the intent of enhancing the predictive accuracy for the next-DTC mode. Due to the constraints imposed by the limited dataset, which preclude the application of larger models, we investigated the potential for further optimization of the Transformer-GRU architecture.

Through experimentation, it was observed that rather than overshadowing the transformer representation by placing a GRU layer immediately after it, synchronizing the two can yield a more enriched context for the output dense layer. This chapter delineates the optimized architecture that enhances the Transformer-GRU framework and presents the associated results, including an ablation study.

Additionally, we examine the implementation of an ensemble model to amalga-

mate the predictions from various individual models, thereby alleviating the inherent limitations of each model when used in isolation and achieving superior overall performance. The ultimate objective is to devise a robust and interpretable predictive maintenance system capable of effectively managing complex sequential dependencies in DTCs, despite the limited availability of data.

7.1 Introduction

The approach introduced in the previous chapter, utilizing a hybrid model combining transformer and Gated Recurrent Unit (GRU) architectures to predict the next DTC in a sequence, demonstrated superior performance compared to existing approaches, achieving a top-5 accuracy of 81.4% in predicting the next DTC event, including its associated features. However, the complex nature of vehicle diagnostics and the potential for further improvements motivated us to explore additional enhancements to this model.

In this Chapter, we have made the following two main contributions:

1. Architectural Refinements: We introduce several modifications to the DTC-TranGru architecture, including better alignment and optimized combination of transformer output with GRU output, introduction of an EOS token, and strategic placement of 1D spatial-dropout layers. These changes enhanced the model's ability to capture complex dependencies in DTC sequences. Our architecture was able to reduce the validation loss to 4.28 and achieve the top-5 accuracy of 82.13%.

2. Ensemble Approach: We develop an ensemble approach that combines our enhanced DTC-GOAT model with two complementary architectures. This combination of models leverages the strengths of different approaches, leading to better top-5 accuracy of 83.15%, which is greater than individual architectures.

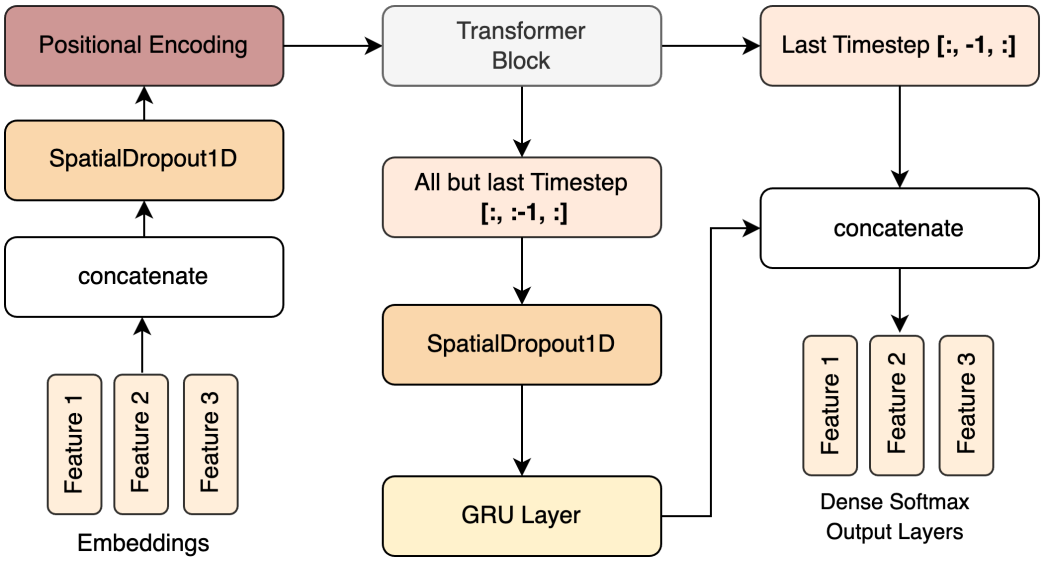


Figure 7.1: Overall architecture of DTC-GOAT. Pre-transformer layers shown on the left side of the figure are common with DTC-TranGru [35]. This includes individual embedding layers for each feature (attribute), followed by concatenation along the time axis, 1D spatial-dropout, and positional encoding. The middle and right part of the figure depicts the true difference between the proposed model and DTC-TranGru, where we pass all but the last timestep (EOS token) from the Transformer output to the GRU layer and concatenate the last transformer timestep with the hidden state of the last timestep from GRU, before passing it to all 3 dense layers. The figure shows that we introduce a 1D spatial-dropout before the GRU layer.

The rest of this chapter is organized as follows. Section 2 details our methodology, including the DTC-GOAT architecture and the ensemble approach. Section 3 presents our experimental setup and results. Section 4 discusses the implications of our findings, and Section 5 concludes the paper with suggestions for future work.

7.2 Methodology

This section provides details of our proposed architecture, which we call DTC-GOAT. We share how our model enhances the DTC-TranGru model by incorporating different architectural changes. Furthermore, we provide details about our ensemble approach for predicting the next DTC event in the sequence. In both contributions, the core

idea remains to predict the next Diagnostic Trouble Code (DTC) event in a sequence but with improved accuracy and robustness.

7.2.1 DTC-GOAT - Building blocks

Our architecture has some common layers with DTC-TranGru, but differs in how these layers are connected, and introduces some new layers along with changes in parameters, enabling increased capacity of the model with better alignments and connections across the layers. Figure 7.1 presents a high-level overview of the new architecture and Algorithm 5 provides implementation details for it.

The subsections ahead describe important blocks in our architecture.

7.2.1.1 Pre-Transformer layers

We employ three separate embedding layers, one for each feature of the DTC event. Every layer has a different embedding size depending on the number of unique classes of that particular feature. We combine these embedding layers across time dimension and pass these to a 1D spatial-dropout layer, which applies the dropout across a time dimension.

Since the transformer layer does not cater for the positional information of the tokens, we apply the positional encoding technique described in the original transformer model [103] to concatenated embeddings, before passing it to the transformer block.

7.2.1.2 Transformer block

We use two transformer encoder layers, each starting with a multi-head attention layer consisting of 3 heads. It is followed by residual connection combining positional encoding (or the previous encoder layer's output) with multi-head attention's output. Afterwards, the output of the residual connection is passed to the first layer-normalization operation, which is followed by a Feed-Forward Network (FFN) block

Algorithm 5 DTC-GOAT

Require: $seq_1 \dots seq_N$
Ensure: DTC event ($feat^1$,$feat^2$,$feat^3$) for $seq_1 \dots seq_N$
  for $epoch \leftarrow 1$ to N **do**
    $f^1_{EMB} \leftarrow EMB(f^1_{OHE})$
    $f^2_{EMB} \leftarrow EMB(f^2_{OHE})$
    $f^3_{EMB} \leftarrow EMB(f^3_{OHE})$
    $f_{EMB} \leftarrow CONCAT(feat^1_{EMB}, feat^2_{EMB}, feat^3_{EMB})$
    $pos\_enc \leftarrow POSITIONAL\_ENCODING(f_{EMB})$
    $pos\_enc \leftarrow 1DSpatialDropout(pos\_enc)$
    $enc \leftarrow pos\_enc$
    for $encoder\_layer \leftarrow 1$ to N **do**
      $multihead \leftarrow multihead_n(enc)$
      $l\_norm\_1 \leftarrow Layer\_Norm(pos\_enc + multihead)$
      $ffn\_l1 \leftarrow Dense(l\_norm\_1)$
      $ffn\_l2 \leftarrow Dense(ffn\_l1)$
      $l\_norm\_2 \leftarrow Layer\_Norm(ffn\_l2 + l\_norm\_1)$
      $enc \leftarrow l\_norm\_2$
    end for
    $LAST\_TIMESETP \leftarrow enc[:, -1, :]$
    $ALL\_BUT\_LAST\_TS \leftarrow enc[:, : -1, :]$
    $spatial\_dropout \leftarrow 1DSpatialDropout(ALL\_BUT\_LAST\_TS)$
    $GRU\_OUTPUT \leftarrow GRU(spatial\_dropout)$
    $CONCATENATED \leftarrow CONCAT([GRU\_OUTPUT, LAST\_TS])$
    $feat\_out^1 \leftarrow Dense(CONCATENATED)$
    $feat\_out^2 \leftarrow Dense(CONCATENATED)$
    $feat\_out^3 \leftarrow Dense(CONCATENATED)$
    $loss \leftarrow \text{Loss}(feat\_out^1, feat\_out^2, feat\_out^3)$
    OptimizeParameters($loss$)
  end for

consisting of two dense layers, the first one with 256 dimensions and the second one with 52 dimensions.

After the FFN block, we have a residual connection between the output of the first layer normalization operation and the second FFN dense layer, followed by the second and final normalization layer. Figure 7.2 presents a detailed view of the single encoder layer of the transformer block.

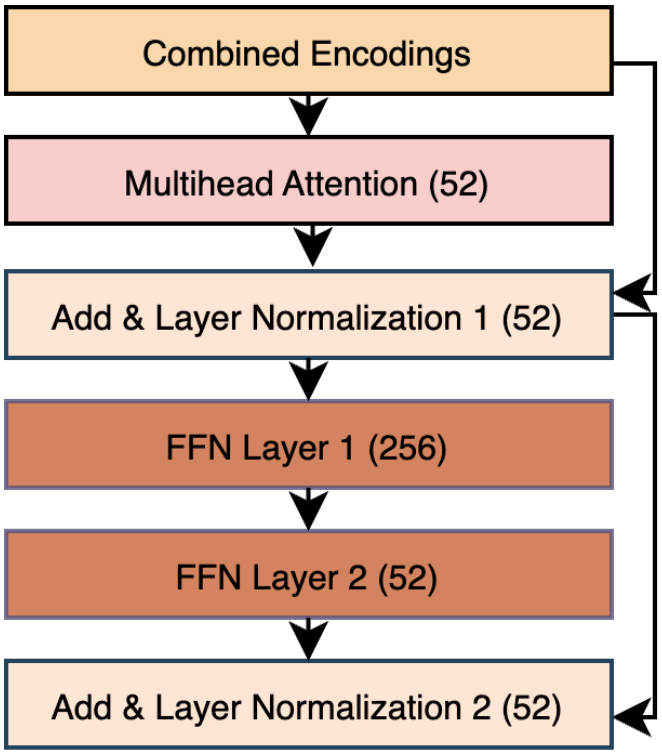


Figure 7.2: Detailed view of transformer encoder layer. The encoder layer receives either a 52-dimensional combined positional encoding vector or the output of the previous encoder layer as an input. This input is passed to the multi-head attention layer and added to the output of the multi-head attention layer to act as a residual connection. This added residual connection is then passed to the first layer normalization operation. After layer normalization, we employ a Feed-Forward Network (FFN) block, which first increases the dimensionality of the input to 256 dimensions before bringing it back to the original size of 52 dimensions.

7.2.1.3 Transformer to GRU Layer

As seen in figure 7.5 as a part of preprocessing, we introduce an end-of-sequence (EOS) token at the end of each feature's sequence at $(N+1)^{th}$ index. In many transformer architectures, the representation associated with the EOS token is argued to entail enough information about all timesteps to be used solely to perform downstream tasks, e.g., classification in BERT [16].

We pass the output of the transformer excluding the last timestep (EOS) to the 1D spatial-dropout first and eventually to the GRU layer.

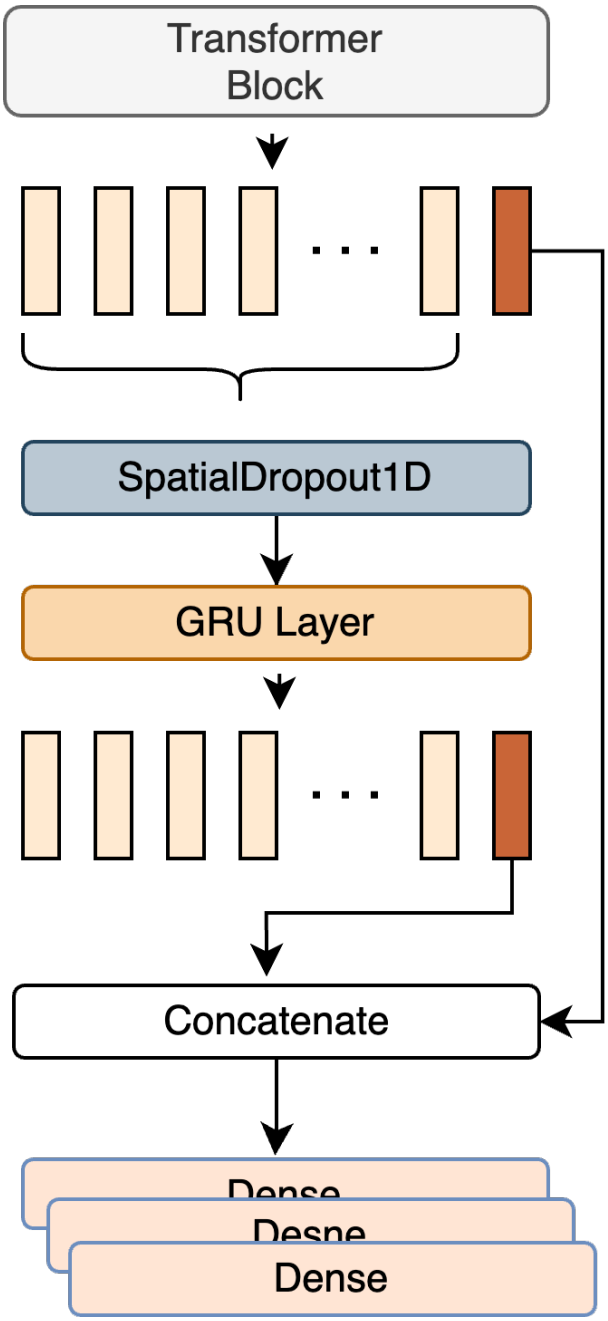


Figure 7.3: This figure shows the detailed view of how the Transformer and the GRU outputs are combined using an EOS token. The last timestep in transformer output corresponds to the EOS token, and we pass all but this timestep of transformer output to the 1D spatial-dropout layer followed by the GRU layer. The last timestep from the transformer is then concatenated with the hidden state of the last timestep from the GRU layer and is then passed to each dense layer for individual features.

7.2.1.4 Combining Transformer and GRU outputs

For the GRU, we used one layer with 256 units. With the input of N timesteps coming from the transformer, ignoring the EOS token, the GRU produces N hidden state vectors. Since the last hidden dimension of the GRU is believed to contain all the essential information from the complete sequence, we use this hidden state and ignore other timesteps from the GRU.

As seen in figure 7.3, we concatenate this last hidden dimension of GRU (256-dimension) with the representation of EOS (52-dimension), which is the last timestep of transformer output, to get a 308-dimensional vector.

7.2.1.5 Dense output layers

This concatenated vector is then passed to the output block. In the output block, we have 3 output dense layers, each corresponding to an individual feature for the DTC event. We use softmax as an activation function in each layer, and the categorical cross-entropy function to calculate the loss. Total loss is calculated by summing the individual feature losses with same weights

$$L(\hat{y}, y) = \sum_{F^i}^{F}(-\sum_{k}^{K_{fi}} y^{(k)} \log(\hat{y^{(k)}})), \tag{7.1}$$

where K_{f^i} represents the number of unique classes in feature (attribute) f_i, F denotes the number of total features, while $\hat{y}$ is the predicted class, and y represents the actual class.

7.2.2 Ensemble of Models

As a second contribution to our work, we propose an ensemble approach [17] for predicting the top-5 accuracy of the next DTC. As shown in figure 7.4, instead of taking the class with the highest probability for each feature from one individual

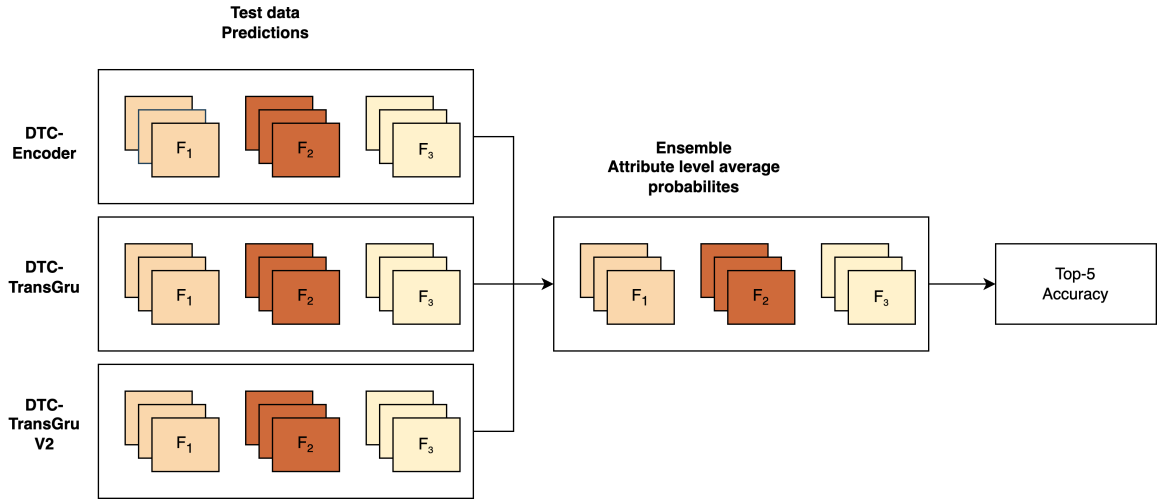


Figure 7.4: We first pass the test dataset to three different models (DTCEncoder, DTC-TranGru, and DTC-GOAT) to get three sets of predictions, and then for each feature, we take the average of probabilities for all class predictions across three models. These average class probabilities are then used to calculate top-5 accuracy.

model, we first take the average of the probabilities from all 3 models, across features. After the average prediction is computed for each feature, we calculate top-5 accuracy, which counts how many times the actual class for each feature has simultaneously existed in the top 5 predicted classes from the model.

Algorithm 6 shows how the feature level average is calculated using predictions with three different models.

Algorithm 6 Ensemble Approach

Require: $test\_pred_1, test\_pred_2, test\_pred_3$
Ensure: Ensemble predictions ($average\_pred$)
  for $f \leftarrow 1$ to F **do**
    for $row \leftarrow 1$ to N **do**
      $average\_pred[f][row] \leftarrow (test\_pred_1[f][row] + test\_pred_2[f][row] + test\_pred_3[f][row])/3$
    end for
  end for

7.3 Experiments and Results

In this section, we will provide details about the data, data preprocessing, and experimental setup for the DTC-GOAT, and share the results of the experiments at the end of the section.

7.3.1 Dataset and Data Preprocessing

We used a dataset provided by *Bosch Automotive*, which consisted of 250k sequences, each corresponding to a unique vehicle and containing a varying number of multivariate DTC fault events.

As a part of preprocessing, the events within each sequence were sorted as per the time when they occurred. As seen in figure 7.5, we form a separate vector of each feature for all sequences to facilitate the application of a separate embedding layer per feature. We also append a special EOS token at the end of each feature sequence.

We trim longer sequences to the last N timesteps, which in our case was 10 timesteps, and pad the shorter sequences with a special token (‘0‘). For the train-test split, we kept $4,750$ sequences for validation purposes, 12,500 as testing data and the rest of $232,750$ sequences were used as training data.

7.3.2 Experimental setup and hyperparameter tuning

We tried multiple hyperparameters and parameters mentioned in table 7.3.2 for our model, and the final parameters were selected with Hyperband [55] hyperparameter tuning method, executed with keras-tuner [74] python library. For the ablation study, apart from comparing with related models, we also compared with modified versions of these models, which were enhanced by adding the 1D spatial-dropout layer before recurrent layers and changing the parameters to the ones obtained in our experiments. The goal was to perform a true comparison and show that the results we achieved

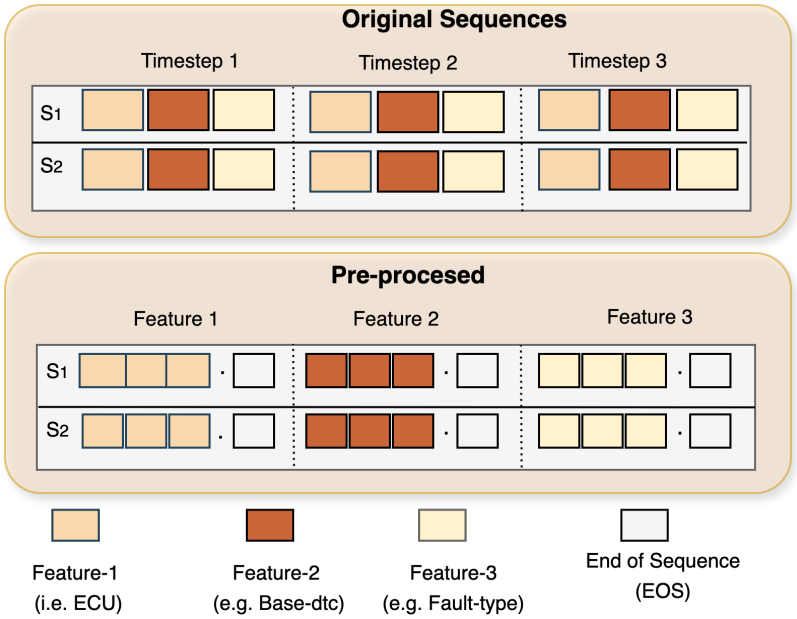


Figure 7.5: The figure shows 2 unique DTC sequences before and after preprocessing. In the preprocessing step, we first separate each feature into separate vectors, keeping the original time order. We restrict the number of DTCs to 3 for the example, but in the original dataset each sequence has at least 5 DTC events. Each feature vector is appended with an end-of-sequence (EOS) token, to reinforce the completion of the sequence and to concatenate the representation of this token with GRU's output.

are not influenced by these optimizations only.

On the model level, the main choice was the learning rate; we used a min value of $1e^{-4}$ and a max value of 0.1 as the learning parameter, and we got the best performance with the value of 0.0005. Another important choice was the value for dropout in 1D spatial-dropout layers in the model, for which we tried the values up to 0.5, but found the best value to be 0.1.

In the transformer block, we experimented with 2 to 5 encoder layers and found two encoder layers to be best performing. We saw that 3 heads performed most optimally in the multi-head attention layer, and 256 dimensions were the best performance for the first layer of the FFN block. Unlike DTC-TranGru, in our case, the GRU layer gave the best result with 256 units.

We used the RELU [69] activation function in the GRU layer and softmax in the

Choice	min	max	final
Learning rate	1e-4	0.1	0.005
Number of heads	1	6	3
Number of encoder layers	1	5	2
FFN Dimension	128	256	256
Feature-1 (module) embedding	4	24	8
Feature-2 (base-dtc) embedding	12	56	32
Feature-3 (fault-byte) embedding	4	32	12
Spatial dropout after embeddings	0.0	0.5	0.1
GRU layer units	96	256	256

Table 7.1: Min and max values of hyperparameters and parameters tried with Hyperband. Some of the important choices include learning rate, number of encoder layers, number of heads in multi-head attention, embedding size and the number of GRU layer units.

output dense layers.

7.3.3 Results

We compare the proposed model with several others, including our first architecture SMFP that introduces the next-DTC prediction problem and implements an LSTM-based architecture utilizing neural embeddings. Another comparison involves our second model (DTCEncoder) that encodes DTC, applying long-attention [61] to GRU outputs and utilizing a dense bottleneck for compact representations. Additionally, we compare our proposed model with a modified small-GPT-2 [84] transformer decoder architecture [81], before comparing it to DTC-TranGru, which employs a GRU immediately after the transformer [103] and uses the GRU output solely in the final layers.

The results of the experiments and comparisons with other architectures can be seen in table 7.3.3. It shows that our model DTC-GOAT was able to achieve the best test top accuracy of 82.13% and the lowest validation loss of 4.28. As a part of the ablation study, we also show that the parameter changes and optimization borrowed from our architecture did improve other architectures slightly, but did not

surpass the metrics of our architecture. We show that allowing the direct influence of transformers representation on the output via concatenation of EOS token with GRU, instead of passing transformer's representation indirectly via GRU, results in an increment of model capacity. The concatenation of the last hidden state of the GRU and the transformer's output for the EOS token's index acts similarly to a skip connection and provides a better flow of information through the network.

For the ensemble approach, it can be seen in the table 7.3.3 that the top-5 accuracy achieved by the ensemble model is 83.15%, which is higher than the accuracies achieved by individual models. This reaffirms our assumption that in the situation where the dataset is limited and having a large model isn't possible, we can use multiple small models to get better results. Since each model learns different features and focuses on different latent patterns, taking the average probability for each feature has the potential to reduce uncertainty and hence boost the overall top-5 accuracy.

Architecture	Validation Loss	Top-5 Test Accuracy
SMFP	4.50	76.15%
SMFP (Opt)	4.49	76.23%
DTCEncoder	4.36	79.21%
DTCEncoder (Opt)	4.35	79.32%
Transformer-Decoder	7.6	40%
Transformer-Decoder (Opt)	7.54	41.4%
DTC-TranGru	4.33	81.4%
DTC-TranGru (Opt)	4.32	81.47%
DTC-GOAT	**4.28**	**82.13**%

Table 7.2: Experiment results for DTC-GOAT compared with other models, like SMFP, DTCEncoder, Transformer Decoder model [81], DTC-TranGru. For the ablation study and to analyse whether changes in parameters and the spatial-dropout layer improved the performance alone, we tried enhanced versions of all the compared models by introducing minor optimization changes used in our model. As highlighted by the bold text, DTC-GOAT achieved the best results among all other models

Architecture	Top-5 Test Accuracy
DTCEncoder	79.32%
DTC-TranGru	81.47%
DTC-GOAT	82.13%
Ensemble of Models	**83.15%**

Table 7.3: Top-5 accuracy results from an ensemble of multiple models compared with individual models. Combining predictions from three models boosts the top accuracy, and turns out to be higher than the top-5 accuracy achieved by individual models

7.4 Discussion and Conclusion

This chapter introduces a model incorporating several optimizations and changes to the architecture of Transformer-GRU architecture we introduced in chapter 7 to improve the top-5 accuracy of the next-DTC prediction task. The optimizations we propose include combining the output of the transformer model and the GRU model in a better way using the EOS token. We argue that since the output of the transformer is now reaching separately to the output layer with the help of the EOS token's representation concatenation with the GRU output, instead of being shadowed by GRU, it increases the model capacity and doesn't overfit early as opposed to other models.

We also introduce an ensemble approach of combining multiple models to attain the class probabilities of each feature in DTC predictions, which gives a better top-5 accuracy of 83.15% as compared to the individual participating models in the ensemble approach. We believe that making use of multiple small models, which might all have learned different patterns, is useful to increase the accuracy of the task where data is limited and it is not possible to build a very large model.

Chapter 8

Conclusion

This dissertation presents a comprehensive examination of the issues associated with Next-Event Prediction in Diagnostic Trouble Code (DTC)-based predictive maintenance for vehicles. It diverges from the constraints of traditional methods, which often depend on simplistic algorithms and suffer from limited data availability, by introducing an innovative architecture that utilizes deep learning models to address the intrinsic complexity of DTCs directly.

This study and problem formulation are of significance as the increasing installation of diagnostic modules in vehicles signals a shift in predictive maintenance towards the usage of DTC fault events, along with warranty and repair data. This methodology eliminates the limitation of concentrating on a select few DTCs and proves beneficial even in scenarios where warranty and repair data are lacking.

8.1 Significance

The development of the proposed next DTC prediction framework represents a significant advancement in the field of predictive maintenance. By accurately forecasting upcoming diagnostic trouble codes based on historical fault sequences, the model enables earlier detection of emerging issues and supports more informed maintenance

scheduling. This proactive approach not only enhances vehicle reliability and operational safety but also has the potential to reduce maintenance costs and downtime. As such, the methodology introduced in this work could contribute to a paradigm shift in how predictive maintenance is implemented across automotive and related domains. Furthermore, individual models shared in each chapter provide details on the applicability and comparison of different models, along with suggestion on how to improve the performance of the prediction task in limited dataset scenario

8.2 Contributions

Through our study, we have demonstrated the effectiveness of combining sequential dependencies with high-level representations, as embodied in architectures such as SMFP, DTC-TranGru and DTCEncoder. Our experiments and methodologies enable end-to-end supervised learning, which can leverage state-of-the-art sequential algorithms instead of relying only on simple algorithms, achieving improved accuracy and robustness in predicting the next event with all three attributes per timestep.

The contributions of this dissertation can be summarized as follows:

- **Sequential Next-DTC prediction problem**: In Chapter 4, we introduced a self-supervised deep learning methodology for predicting the next Diagnostic Trouble Code (DTC) event, thereby effectively addressing the intrinsic complexities of DTCs. Our models exploit sequential dependencies through the application of algorithms such as Long Short-Term Memory (LSTM) layers to enable precise predictions of subsequent events encompassing all three attributes per time step.

  Moreover, we proposed top-3 and top-5 accuracy metrics for evaluating model performance, establishing a baseline with the results of the SMFP model introduced in Chapter 4.

- **DTC Representation**: In the inaugural architecture discussed in Chapter 4, we demonstrated the employment of neural embeddings for representing multiple high cardinality attributes within DTC events. The representation of DTC events as dense features facilitated the application of end-to-end self-supervised learning via sequential algorithms.

  We elucidated how utilizing separate embedding layers introduces flexibility in class representation within a single attribute, allowing concatenation to form a comprehensive representation of the entire DTC event. This methodology, presented in Chapter 4, updates embedding weights in conjunction with the overall DTC prediction framework via a joint learning approach, supported by multiple output dense layers corresponding to each attribute of the DTC event.

- **Interpretability and Retrieval**: In the second approach detailed in Chapter 5, we achieved a compact representation of DTC sequences using an attention mechanism alongside a bottleneck dense layer, showcasing various applications for these dense representations. We illustrated how the learned compact DTC sequence representations can assist engineers in retrieving comparable sequences, thereby facilitating the development of repair strategies.

  Additionally, we demonstrated how these representations can be employed for clustering sequences into distinct groups, aiding in the identification of common issues and further refinement of retrieved analogous sequences. We also provided an example of layman's interpretation of the model's decision using attention weights to highlight the significance of each DTC event in the final prediction.

  This approach not only enables varied applications of compact representations but also enhances the DTC prediction's top-5 accuracy benchmark.

- **Hybrid Transformer-Gru model to further Enhance model accuracy**: Due to data limitations and the unavailability of additional data within the timeframe, we explored alternative methodologies requiring minimal data. This

necessitated examining state-of-the-art sequential algorithms such as Transformers. In Chapter 6, we demonstrated that combining a small Transformer with another sequential algorithm can still yield favorable results.

We showed that employing a GRU layer subsequent to a Transformer was effective in our case, outperforming standalone Transformer models.

- **Pushing the Next-DTC prediction accuracy further with architectural improvements and Ensembles**: In Chapter 7, we introduced architectural improvements to the previously discussed transformer-GRU model. We demonstrated that implementing modifications, such as combining the Transformer representation with GRU without overshadowing, enhances model performance. These incremental changes are crucial in scenarios with limited datasets where each minor improvement matters.

  Consistent with this theme, we presented a straightforward ensemble approach leveraging multiple DTC prediction models to average each attribute's top-class prediction, thereby enhancing the overall top-5 prediction accuracy.

8.3 Limitations

A primary limitation of this study is the size of the dataset available for experimentation. The objective of this doctoral research was to evaluate the proposed algorithms on a real-time dataset sourced from connected vehicles, rather than on datasets extracted from workshop sessions, which are also constrained by the number of diagnostic trouble code (DTC) sequences.

Beyond the total size of the dataset, i.e., the number of sequences, there was significant variability in the DTC events among individual vehicles. As detailed in Chapter 4, an attempt was made to employ a windowing approach to leverage longer sequences to augment the dataset's size; however, due to the adequate presence of sequences with extensive DTC events, the strategy proved minimally effective.

Another limitation of this work is the inability to obtain warranty and repair data for the vehicles, necessitating the recasting of the problem into a self-supervised learning framework. This limitation likely impeded the potential benefits of cross-comparison with actual repairs, and it would have been insightful to identify which DTCs were triggered and whether they resulted in actual failures. It is posited that further advancements could have been achieved if there had been an opportunity to apply these methodologies to other DTC events. However, the lack of access to additional datasets from the company and public sources constrained the full realization of this research's value.

8.4 Future work

Despite this dissertation yielding several useful models, published papers, and a patent application, time constraints suggest that there remains significant scope for further work, particularly once additional datasets become available. Potential avenues for future exploration are outlined below.

8.4.1 Changes for large and imperfect datasets

In more extensive datasets, which are not curated specifically for research purposes like ours, issues such as class imbalance in the DTCs may arise. It would be beneficial to investigate metrics (e.g., F1 score) and loss functions (e.g., focal loss [56]) that are more resilient to imbalanced datasets. Furthermore, with the availability of longer DTC sequences, it would be worthwhile to experiment with larger sequence sizes to assess how the proposed algorithms perform with increased contextual information.

8.4.2 Combining warranty and repair data to enhance supervised learning performance

It may be advantageous to integrate or repurpose the representations acquired from a self-supervised learning approach, such as next-DTC prediction, to improve the performance of DTC classification tasks, including distinguishing between faulty and non-faulty sequences utilizing warranty and repair data.

8.4.3 Predicting when will the next DTC occur

This thesis predominantly addresses the 'What' aspect of the next-DTC prediction problem. Another potential improvement lies in incorporating the 'When' aspect into predictions by including features such as time and mileage at the event's occurrence. Although precise prediction of event timing was not initially required in our use case, this addition could be instrumental in forecasting the necessity for future maintenance.

8.5 Personal Reflection

This PhD commenced in February 2020 and was among the inaugural cohort of Industrial PhDs under the EIT-City Industrial PhD initiative [14]. The inception of this PhD occurred during a particularly challenging period marked by the simultaneous events of Brexit and the COVID-19 pandemic. Due to Brexit aftereffects, EIT Digital later left the project, which produced some funding issues, but with the support of supervisors, our research lab CITAI, and the City St George's University of London, the funding was covered.

The industrial collaborator for this project, Bosch Automotive (currently an ETAS subunit), aimed to implement the outcomes of this research on data obtained from connected vehicles in real-time. However, the advent of COVID-19 presented chal-

lenges related to data acquisition, customer retention, and data sharing frameworks, which were further complicated by Brexit. Despite these impediments and the limited availability of data, the research team succeeded in reframing the problem through a novel approach. This method was deemed unique by Bosch's internal patent advisory team and attracted several proof of concept (POC) requests from other clients. Initially, there was strong anticipation of acquiring warranty and repair data from customers to support the analysis of Diagnostic Trouble Codes (DTCs) and enhance the problem framing with more actionable feedback to refine the algorithms. Unfortunately, the warranty and repair data were only made available towards the conclusion of this PhD project. Consequently, Bosch has initiated another studentship aimed at exploring how the existing research can be leveraged alongside the newly acquired warranty and repair data.

At the outset of the project, Bosch employed aggregation-based metrics for the development of DTC co-occurrence and prediction models, including some algorithms that did not account for sequential dependencies, such as association rule mining [114]. Beyond advancing Bosch's existing models, the objective of this PhD was to develop artificial intelligence or machine learning algorithms. Therefore, the focus shifted towards utilizing DTC data without the aid of feedback indicators, such as warranty and repair data, in a novel manner diverging from current count/aggregation-based methodologies and association mining.

The initial research detailed in Chapter 4 involved developing a methodology to represent multivariate Diagnostic Trouble Code (DTC) events characterized by high cardinality, or a significant number of unique classes for each attribute. We introduced the use of neural embedding techniques to create dense representations for each DTC attribute class, thereby facilitating the application of machine learning algorithms that require numerical data formats. This representation method allowed the implementation of a self-supervised learning strategy, wherein the presence of labels is unnecessary. Given the absence of feedback labels in the form of warranty

and repair data, the self-supervised learning paradigm was deemed more suitable. We employed sequential algorithms such as recurrent neural networks (e.g., Long Short-Term Memory Networks, LSTMs) on top of the learned neural embeddings to predict the three attributes of the DTC event at each time step. This approach was published in IEEE ICMLA [34], and a patent application [44] has been filed for this innovation.

Subsequently, we examined various methodologies to enhance this model, focusing on both accuracy and applicability. As detailed in Chapter 5, we investigated the development of a method that not only provides a concise representation of individual Diagnostic Trouble Codes (DTCs) but also encapsulates the entire sequence. This comprehensive representation can be utilized in various applications, such as expediting the retrieval of similar fault sequences. This capability assists engineers in determining if any other vehicle has previously exhibited the same pattern. Furthermore, the attention mechanism integrated into the model enables engineers to interpret which faults significantly influenced the mode's decision-making process. This approach is more comprehensible for engineers compared to existing scientific methods of interpretability, such as SHAP [57].

To enhance model accuracy, we integrated the latest state-of-the-art models, such as transformers, with recurrent models like GRUs, resulting in hybrid models that further improved accuracy. In the model presented in Chapter 6, we demonstrated that utilizing a large SOTA model in its entirety is not advantageous for scenarios with data limitations. However, employing a significantly smaller version of these models can still positively impact performance.

In the last work, shared as Chapter 7, we demonstrated that model performance can be enhanced by optimizing the existing architecture through a thorough understanding of its limitations and the implementation of ingenious architectural improvements. Additionally, we illustrated that leveraging the performance of various models through a straightforward ensemble method can effectively maximize data utility for performance advancement.

In addition to attracting interest from Bosch customers, we have also observed researchers drawing inspiration from our work and pursuing a similar direction in using a sequential approach to predict DTC events. This is evidenced by a citation from Amazon [115], where they propose different techniques, such as time-aware event sequence embedding, for event log-based predictive maintenance. Researchers from Augsburg [65], in collaboration with BMW, are also employing sequential algorithms inspired by our use of transformers and have cited both the SMFP and DTC-Transgru models from our work.

It is our assessment that the disruptions caused by the COVID-19 pandemic, Brexit, and the departure of the EIT led to challenges in the research process. However, considering that this PhD was industry-oriented, we succeeded in developing models that garnered substantial interest from customers. These models were designed with a focus on addressing the specific problem at hand, rather than constructing overly complex solutions which would not have served the needs of our industrial partner effectively.

In conclusion, the Industrial PhD program provided a distinct and valuable experience for both the student and the supervisor, owing to its fundamental differences from a traditional research doctorate. While emphasizing the innovation of solutions in the DTC prediction domain, it was equally critical for the candidate to develop solutions that would be beneficial to the industrial partner. We contend that there is significant potential for the enhancement and expansion of these models, and further reflections on this matter are presented in the subsequent section.

Bibliography

[1] Mahfuz Alam, Md Rafiqul Islam, and Sanjib Kumar Shil. "AI-Based predictive maintenance for US manufacturing: reducing downtime and increasing productivity". In: *International Journal of Advanced Engineering Technologies and Innovations* 1.01 (2023), pp. 541–567.

[2] Md Zahangir Alom et al. "Deep Versus Wide Convolutional Neural Networks for Object Recognition on Neuromorphic System". In: (2018). arXiv: 1802.02608 [cs.CV].

[3] Apache Software Foundation. *Apache Spark*. Version 2.4.6. URL: https://spark.apache.org/.

[4] Fabio Arena et al. "Predictive Maintenance in the Automotive Sector: A Literature Review". In: *Mathematical and Computational Applications* 27.1 (2022). ISSN: 2297-8747. DOI: 10.3390/mca27010002. URL: https://www.mdpi.com/2297-8747/27/1/2.

[5] Oren Barkan and Noam Koenigstein. "Item2vec: neural item embedding for collaborative filtering". In: *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2016, pp. 1–6.

[6] Ernnie Illyani Basri et al. "Preventive maintenance (PM) planning: a review". In: *Journal of quality in maintenance engineering* 23.2 (2017), pp. 114–143.

[7] Mattia Beretta et al. "An Ensemble Learning Solution for Predictive Maintenance of Wind Turbines Main Bearing". In: *Sensors* 21.4 (2021). ISSN: 1424-

8220. DOI: 10.3390/s21041512. URL: https://www.mdpi.com/1424-8220/21/4/1512.

[8] Bosch. *Bosch*. URL: https://www.bosch.com/.

[9] John Bridle. "Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper_files/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf.

[10] Kunru Chen et al. "Predicting air compressor failures using long short term memory networks". In: *Progress in Artificial Intelligence: 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, September 3–6, 2019, Proceedings, Part I 19*. Springer. 2019, pp. 596–609.

[11] Yanqing Chen et al. "The expressive power of word embeddings". In: *arXiv preprint arXiv:1301.3226* (2013).

[12] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[13] François Chollet. *open-source library that provides a Python interface for artificial neural networks*. https://keras.io/api/layers/recurrent_layers/lstm/.

[14] CitAi. *Industrial Doctoral Training Program*. URL: https://cit-ai.net/Industrial-PhD.html.

[15] Narjes Davari et al. "Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry". In: *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. 2021, pp. 1–10. DOI: 10.1109/DSAA53316.2021.9564181.

[16] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *North American Chapter of the Association for Computational Linguistics*. 2019. URL: https://api.semanticscholar.org/CorpusID:52967399.

[17] Xibin Dong et al. "A survey on ensemble learning". In: *Frontiers of Computer Science* 14 (2020), pp. 241–258.

[18] Philipp Dufter, Martin Schmitt, and Hinrich Schütze. "Position Information in Transformers: An Overview". In: *Computational Linguistics* 48 (June 2022), pp. 1–31. DOI: 10.1162/coli_a_00445.

[19] Jeffrey L. Elman. "Finding Structure in Time". In: *Cognitive Science* 14.2 (1990), pp. 179–211. DOI: https://doi.org/10.1207/s15516709cog1402_1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

[20] Diogo R. Ferreira, Teresa Scholz, and Rune Prytz. "Importance Weighting of Diagnostic Trouble Codes for Anomaly Detection". In: *Machine Learning, Optimization, and Data Science*. Ed. by Giuseppe Nicosia et al. Cham: Springer International Publishing, 2020, pp. 410–421. ISBN: 978-3-030-64583-0.

[21] William J Fleming. "Overview of automotive sensors". In: *IEEE sensors journal* 1.4 (2001), pp. 296–308.

[22] Moa Fransson and Lisa Fåhraeus. *Finding Patterns in Vehicle Diagnostic Trouble Codes : A data mining study applying associative classification*. 2015. URL: %5Curl%7Bhttp://uu.diva-portal.org/smash/get/diva2:828052/FULLTEXT01.pdf%7D.

[23] Yarin Gal. *Uncertainty in Deep Learning (PhD Thesis)*. http://mlg.eng.cam.ac.uk/yarin/blog_2248.html. 2015.

[24] Gabriella Galonja. "Improving vehicle diagnostic solutions for heavy-duty vehicles through machine learning". PhD thesis. Uppsala universitet, 2020. URL: %5Curl%7Bhttps://odr.chalmers.se/items/638611c9-4dfd-4f5d-bf5c-03e11ce657ea%7D.

[25] F. A. Gers, D. Eck, and J. Schmidhuber. "Applying LSTM to Time Series Predictable Through Time-Window Approaches". In: (2001). Ed. by Georg Dorffner, pp. 669–676.

[26] Flavio Giobergia et al. "Mining Sensor Data for Predictive Maintenance in the Automotive Industry". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. Turin, Italy: IEEE, 2018, pp. 351–360. DOI: 10.1109/DSAA.2018.00046.

[27] Chew Cheik Goh et al. "Real-time in-vehicle air quality monitoring system using machine learning prediction algorithm". In: *Sensors* 21.15 (2021), p. 4956.

[28] Hardik A. Gohel et al. "Predictive maintenance architecture development for nuclear infrastructure using machine learning". In: *Nuclear Engineering and Technology* 52.7 (2020), pp. 1436–1442. DOI: https://doi.org/10.1016/j.net.2019.12.029. URL: https://www.sciencedirect.com/science/article/pii/S1738573319306783.

[29] Elliott Gordon-Rodríguez et al. "Uses and Abuses of the Cross-Entropy Loss: Case Studies in Modern Deep Learning". In: *ArXiv* abs/2011.05231 (2020). URL: https://api.semanticscholar.org/CorpusID:226290190.

[30] Margherita Grandini, Enrico Bagli, and Giorgio Visani. "Metrics for multi-class classification: an overview". In: *arXiv preprint arXiv:2008.05756* (2020).

[31] Juan Guerrero-Ibáñez, Sherali Zeadally, and Juan Contreras-Castillo. "Sensor technologies for intelligent transportation systems". In: *Sensors* 18.4 (2018), p. 1212.

[32] Jie Gui et al. “A Survey on Self-supervised Learning: Algorithms, Applications, and Future Trends”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[33] Özlem Güven and Hasan Şahin. “Predictive Maintenance Based On Machine Learning In Public Transportation Vehicles”. In: *Mühendislik Bilimleri ve Araştırmaları Dergisi* 4.1 (2022), pp. 89–98.

[34] AB 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)*. Pasadena, CA, USA: IEEE, 2021.

[35] Abdul Basit Hafeez, Eduardo Alonso, and Atif Riaz. “DTC-TranGru: Improving the performance of the next-DTC Prediction Model with Transformer and GRU”. In: *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*. SAC ’24. Avila, Spain: Association for Computing Machinery, 2024, pp. 927–934. ISBN: 9798400702433. DOI: 10.1145/3605098.3635962. URL: https://doi.org/10.1145/3605098.3635962.

[36] Abdul Basit Hafeez, Eduardo Alonso, and Atif Riaz. “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)*. Nassau, Bahamas: IEEE, 2022, pp. 519–524. DOI: 10.1109/ICMLA55696.2022.00085.

[37] Hadia Hameed, Suleman Mazhar, and Naufil Hassan. “Real-time road anomaly detection, using an on-board data logger”. In: *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*. IEEE. 2018, pp. 1–5.

[38] John A Hartigan and Manchek A Wong. “Algorithm AS 136: A k-means clustering algorithm”. In: *Journal of the royal statistical society. series c (applied statistics)* 28.1 (1979), pp. 100–108.

[39] HM Hashemian. "Wireless sensors for predictive maintenance of rotating equipment in research reactors". In: *Annals of Nuclear Energy* 38.2-3 (2011), pp. 665–680.

[40] Trevor Hastie et al. "Overview of supervised learning". In: *The elements of statistical learning: Data mining, inference, and prediction* (2009), pp. 9–41.

[41] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: (2012). arXiv: 1207.0580 [cs.NE].

[42] Hochreiter-Sepp et al. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[43] Victoria Hodge and Jim Austin. "A survey of outlier detection methodologies". In: *Artificial intelligence review* 22 (2004), pp. 85–126.

[44] Juergen Homung et al. "Fault Prediction for Machines". GB2612362 (A). Nov. 1, 2021.

[45] Yingping Huang et al. "Bayesian belief network based fault diagnosis in automotive electronic systems". In: *Proceeding of 8th International Symposium on Advanced Vehicle Control*. Citeseer. 2006, pp. 469–473.

[46] Nadeem Iftikhar et al. "Outlier Detection in Sensor Data using Ensemble Learning". In: *Procedia Computer Science* 176 (2020). Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 24th International Conference KES2020, pp. 1160–1169. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.09.112. URL: https://www.sciencedirect.com/science/article/pii/S1877050920320123.

[47] Magnus Jansson. "Fault isolation utilizing Bayesian networks". In: *Master's thesis. Royal Institute of Technology, KTH, Stockholm, Sweden* (2004).

[48] Stanisław Jastrzebski et al. "Residual connections encourage iterative inference". In: *arXiv preprint arXiv:1710.04773* (2017).

[49] Ameeth Kanawaday and Aditya Sane. "Machine learning for predictive maintenance of industrial machines using IoT sensor data". In: *2017 8th IEEE international conference on software engineering and service science (ICSESS)*. IEEE. 2017, pp. 87–90.

[50] Young Hyun Kim and Dae Cheol Ko. "Study on techniques for analyzing vehicle fire cases using DTC". In: *Transactions of the Korean Society of Automotive Engineers* 31.2 (2023), pp. 83–88.

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

[52] Rajalakshmi Krishnamurthi et al. "An overview of IoT sensor data processing, fusion, and analysis techniques". In: *Sensors* 20.21 (2020), p. 6076.

[53] Roshan Kumari and Saurabh Kr Srivastava. "Machine learning: A review on binary classification". In: *International Journal of Computer Applications* 160.7 (2017).

[54] Uichin Lee and Mario Gerla. "A survey of urban vehicular sensing platforms". In: *Computer networks* 54.4 (2010), pp. 527–544.

[55] Lisha Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52. URL: `http://jmlr.org/papers/v18/16-558.html`.

[56] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.2 (2020), pp. 318–327. DOI: `10.1109/TPAMI.2018.2858826`.

[57] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable ai: A review of machine learning interpretability methods". In: *Entropy* 23.1 (2020), p. 18.

[58] Ting Liu et al. "An investigation of practical approximate nearest neighbor algorithms". In: *Advances in neural information processing systems* 17 (2004).

[59] H. P. Luhn. "The Automatic Creation of Literature Abstracts". In: 2.2 (1958), pp. 159–165. URL: http://www.research.ibm.com/journal/rd/022/luhn.pdf.

[60] Malte Lukas and Daniel P Anderson. "Machine and lubricant condition monitoring for extended equipment lifetimes and predictive maintenance at power plants". In: *Proceedings of Power-Gen International, Jakarta, Indonesia* (1996).

[61] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. DOI: 10.48550/ARXIV.1508.04025. URL: https://arxiv.org/abs/1508.04025.

[62] Laurens van der Maaten and Geoffrey Hinton. "Viualizing data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[63] Laurens van der Maaten, Eric Postma, and H. Herik. "Dimensionality Reduction: A Comparative Review". In: *Journal of Machine Learning Research - JMLR* 10 (Jan. 2007).

[64] Anqi Mao, Mehryar Mohri, and Yutao Zhong. "Cross-entropy loss functions: theoretical analysis and applications". In: *Proceedings of the 40th International Conference on Machine Learning*. ICML'23. Honolulu, Hawaii, USA: JMLR.org, 2023.

[65] Hugo Math, Rainer Lienhart, and Robin Schön. "Harnessing Event Sensory Data for Error Pattern Prediction in Vehicles: A Language Model Approach". In: *arXiv preprint arXiv:2412.13041* (2024).

[66] Andrew Mccallum and Kamal Nigam. "A Comparison of Event Models for Naive Bayes Text Classification". In: *Work Learn Text Categ* 752 (May 2001).

[67] RK Mobley. "An Introduction to Predictive Maintenance". In: *Elsevier Science google schola* 2 (2002), pp. 485–520.

[68] Marek Moleda et al. “From Corrective to Predictive Maintenance—A Review of Maintenance Approaches for the Power Industry”. In: *Sensors* 23.13 (2023). ISSN: 1424-8220. DOI: 10.3390/s23135970. URL: https://www.mdpi.com/1424-8220/23/13/5970.

[69] Vinod Nair and Geoffrey Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair”. In: vol. 27. June 2010, pp. 807–814.

[70] Srikanth Namuduri et al. “Deep learning methods for sensor based predictive maintenance and future perspectives for electrochemical sensors”. In: *Journal of The Electrochemical Society* 167.3 (2020), p. 037552.

[71] Anvardh Nanduri and Lance Sherry. “Anomaly detection in aircraft data using Recurrent Neural Networks (RNN)”. In: *2016 Integrated Communications Navigation and Surveillance (ICNS)*. 2016, pp. 5C2-1-5C2–8. DOI: 10.1109/ICNSURV.2016.7486356.

[72] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. “A review on the attention mechanism of deep learning”. In: *Neurocomputing* 452 (2021), pp. 48–62.

[73] Andreas Nordberg. *Evaluation of Neural Networks for Predictive Maintenance: A Volvo Penta Study*. 2021.

[74] Tom O’Malley et al. *KerasTuner*. 2019. URL: https://github.com/keras-team/keras-tuner.

[75] Chakradhara Panda and Tilak Raj Singh. “ML-based vehicle downtime reduction: A case of air compressor failure detection”. In: *Engineering Applications of Artificial Intelligence* 122 (2023), p. 106031. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2023.106031. URL: https://www.sciencedirect.com/science/article/pii/S0952197623002154.

[76] Seyoung Park et al. “Unsupervised and non-parametric learning-based anomaly detection system using vibration sensor data”. In: *Multimedia Tools Appl.* 78.4

(Feb. 2019), pp. 4417–4435. ISSN: 1380-7501. DOI: 10.1007/s11042-018-5845-4. URL: https://doi.org/10.1007/s11042-018-5845-4.

[77] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, III–1310–III–1318.

[78] Martin Pech, Jaroslav Vrchota, and Jiří Bednář. "Predictive Maintenance and Intelligent Sensors in Smart Factory: Review". In: *Sensors* 21.4 (2021). ISSN: 1424-8220. DOI: 10.3390/s21041470. URL: https://www.mdpi.com/1424-8220/21/4/1470.

[79] Pham et al. "Recurrent Neural Network for Classifying of HPC Applications". In: Apr. 2019, p. 15. DOI: 10.22360/springsim.2019.hpc.015.

[80] Parivash Pirasteh et al. "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. Melbourne, VIC, Australia: Association for Computing Machinery, 2019. ISBN: 9781450362962.

[81] H. Poljo. "Transformer decoder as a method to predict diagnostic trouble codes in heavy commercial vehicles". MA thesis. 2021.

[82] Ekaterina Poslavskaya and Alexey Korolev. "Encoding categorical data: Is there yet anything'hotter'than one-hot encoding?" In: *arXiv preprint arXiv:2312.16930* (2023).

[83] Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. "A comparative study of categorical variable encoding techniques for neural network classifiers". In: *International journal of computer applications* 175.4 (2017), pp. 7–9.

[84] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[85] Deepa Saibannavar, Mallikarjun M Math, and Umakant Kulkarni. "A survey on on-board diagnostic in vehicles". In: *International Conference on Mobile Computing and Sustainable Informatics: ICMCSI 2020*. Springer. 2021, pp. 49–60.

[86] Sule Selcuk. "Predictive maintenance, its implementation and latest trends". In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 231.9 (2017), pp. 1670–1679.

[87] Erhan Sezerer and Selma Tekir. "A survey on neural word embeddings". In: *arXiv preprint arXiv:2110.01804* (2021).

[88] Erhan Sezerer and Selma Tekir. "A survey on neural word embeddings". In: *arXiv preprint arXiv:2110.01804* (2021).

[89] Uferah Shafi et al. "Vehicle Remote Health Monitoring and Prognostic Maintenance System". In: *Journal of Advanced Transportation* 2018 (Jan. 2018), pp. 1–10. DOI: 10.1155/2018/8061514.

[90] Abdulrahim Shamayleh, Mahmoud Awad, and Jumana Farhat. "IoT based predictive maintenance management of medical equipment". In: *Journal of medical systems* 44.4 (2020), p. 72.

[91] Karamjit Singh, Gautam Shroff, and Puneet Agarwal. "Predictive reliability mining for early warnings in populations of connected machines". In: *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 2015, pp. 1–10. DOI: 10.1109/DSAA.2015.7344806.

[92] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*. 2012. arXiv: 1206.2944 [stat.ML].

[93] Vijender Kumar Solanki and Rohit Dhall. "An IoT based predictive connected car maintenance approach". In: (2017).

[94] Spotify. *Annoy*. URL: https://github.com/spotify/annoy.

[95] S. Subramaniam et al. “Online outlier detection in sensor data using non-parametric models”. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*. VLDB ’06. Seoul, Korea: VLDB Endowment, 2006, pp. 187–198.

[96] Anant Subramanian et al. “Spine: Sparse interpretable neural embeddings”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[97] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL].

[98] Jittiwut Suwatthikul, Ross McMurran, and R Peter Jones. “Automotive network diagnostic systems”. In: *2006 International Symposium on Industrial Embedded Systems*. IEEE. 2006, pp. 1–4.

[99] Mikolov Tomas et al. “Efficient Estimation of Word Representations in Vector Space”. In: (Jan. 2013), pp. 1–12.

[100] Denis Torgunov et al. “Vehicle Warranty Claim Prediction from Diagnostic Data Using Classification”. In: *Advances in Computational Intelligence Systems*. Ed. by Zhaojie Ju et al. Cham: Springer International Publishing, 2020, pp. 483–492. ISBN: 978-3-030-29933-0.

[101] Alan Turnbull and James Carroll. “Cost benefit of implementing advanced monitoring and predictive maintenance strategies for offshore wind farms”. In: *Energies* 14.16 (2021), p. 4922.

[102] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[103] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[104] Linda Virkkala and Johanna Haglund. "Modelling of patterns between operational data, diagnostic trouble codes and workshop history using big data and machine learning". PhD thesis. Uppsala universitet, 2016. URL: %5Curl%7Bhttps://www.diva-portal.org/smash/get/diva2:909003/FULLTEXT01.pdf%7D.

[105] Kilian Vos et al. "Vibration-based anomaly detection using LSTM/SVM approaches". In: *Mechanical Systems and Signal Processing* 169 (2022), p. 108752. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2021.108752. URL: https://www.sciencedirect.com/science/article/pii/S0888327021010682.

[106] Chuliang Wei et al. "Real-time train wheel condition monitoring by fiber Bragg grating sensors". In: *International Journal of Distributed Sensor Networks* 8.1 (2011), p. 409048.

[107] Shaomin Wu. "Warranty data analysis: A review". In: *Quality and Reliability Engineering International* 28.8 (2012), pp. 795–805.

[108] Jingjing Xu et al. "Understanding and improving layer normalization". In: *Advances in neural information processing systems* 32 (2019).

[109] Zi Yin and Yuanyuan Shen. "On the Dimensionality of Word Embedding". In: (2018). arXiv: 1812.04224 [cs.LG].

[110] Jiqiang Zhang et al. "Fault diagnosis of bearings based on deep separable convolutional neural network and spatial dropout". In: *Chinese Journal of Aeronautics* 35.10 (2022), pp. 301–312.

[111] Minghu Zhang, Xin Li, and Lili Wang. "An Adaptive Outlier Detection and Processing Approach Towards Time Series Sensor Data". In: *IEEE Access* 7 (2019), pp. 175192–175212. DOI: 10.1109/ACCESS.2019.2957602.

[112] Zhongju Zhang and Pengzhu Zhang. "Seeing around the corner: an analytic approach for predictive maintenance using sensor data". In: *Journal of Man-*

agement Analytics 2.4 (2015), pp. 333–350. DOI: 10.1080/23270012.2015.1086704. URL: https://doi.org/10.1080/23270012.2015.1086704.

[113] Pushe Zhao et al. "Advanced correlation-based anomaly detection method for predictive maintenance". In: *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 2017, pp. 78–83. DOI: 10.1109/ICPHM.2017.7998309.

[114] Qiankun Zhao and Sourav S Bhowmick. "Association rule mining: A survey". In: *Nanyang Technological University, Singapore* 135 (2003), p. 18.

[115] Yun Zhou et al. "Deep sequence modeling for event log-based predictive maintenance". In: (2023). URL: https://www.amazon.science/publications/deep-sequence-modeling-for-event-log-based-predictive-maintenance.